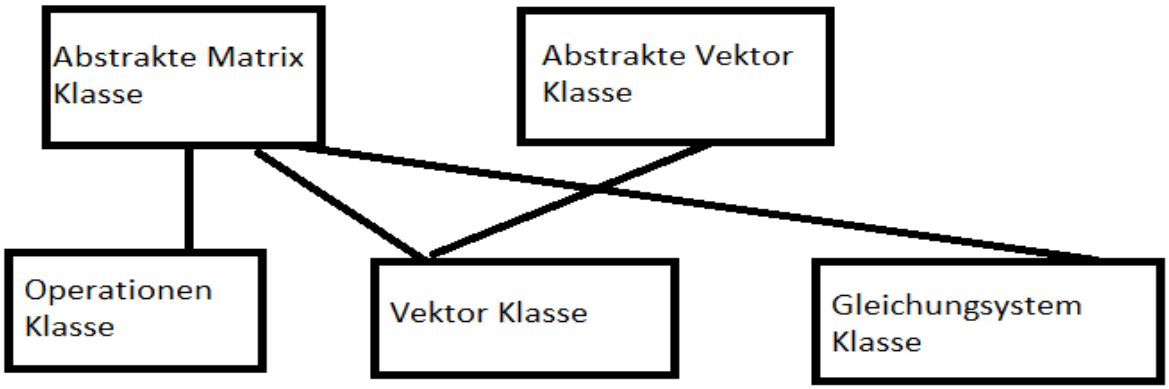
Ich habe in meiner Hausaufgabe 3 template Matrix Klasse verwirklicht. Mein Ziel war die verschiedenen Gebiete den Matrizen vorzustellen. Meine Klassenhierarchie sieht so aus:



Ich habe 2 abstrakte Klasse benutzt damit man leicht dieses Programm erweitern kann. Also wenn man noch ein paar Klasse- die eine Beziehung mit Matrizen oder Vektoren hat- verwirklichen möchte, ist es nicht so schwierig, als in Allgemein. Diese abstrakten Klassen können wir nicht instanziieren. In diesen Klassen habe ich über die Konstruktoren und Destruktoren geschrieben. Beide Klassen haben auch ein „pure virtual print“ Funktion, also, wenn wir dieses Programm erweitern möchten, müssen wir diese Funktion schrieben. Ich denke, dass es „in main“ leichter ist << operator zu benutzen (bei allen Klassen habe ich << überladet.) Es ist empfehlenswert diese Funktion zu haben. (während des Programms etwas ausschreiben usw.) Es gibt keine Funktionen mehr in diesen zwei abstrakten Klassen.

In diesem Programm sind die Klassen template Klassen, deshalb habe ich alle Funktionen in Header geschrieben. In diesem Fall (template Klasse) muss man in „main“ angeben, mit welchen Typen man arbeiten möchte. Leider ist es nicht immer gut einige Typen zu benutzen aber später werde ich darüber schreiben.
Bei den Arrays habe ich dynamische Speicher benutzt.:

```
arr = new type*[m]; //m: Anzahl der Zeile
for (int i = 0; i < m; ++i) {
    arr[i] = new type[n]; // n: Anzahl der Spalte
}
```

Und der Destruktor sieht so aus:

```
virtual ~absmatrix()=0 {
    for (int i = 0; i < m; ++i) {
        delete[] arr[i];
    }
    delete[] arr;
}
```

Erste Klasse ist eine „Operator Klasse“, also es kann Matrix Operationen durchführen. Diese Klasse vererbt die „abstrakte Matrix“ Klasse. In dieser Klasse habe ich viele Operator überladet. Der Benutzer muss nur mit den Operation Zeichnen eingeben, was er ausrechnen möchte, also muss man keine komplizierte Funktion rufen. In dieser Klasse befinden sich „friend“ Funktionen, weil diese Funktionen 2 Parameter haben. Bei der Multiplizierung der Matrizen muss der Benutzer auf die Größe der Matrizen achten. In diesem Programm habe ich „try-catch“ benutzt, deshalb ist es nicht so

großes Problem, wenn der Benutzer etwas falsch macht. (Destruktoren werden automatisch nach der catch geruft.) Die Matrizen können wir nur in der Datei eingeben. (einzelweise braucht man mehr Zeit und es ist unbequem). Die Variablen und das Array, das die Matrix speichert sind private. Es ist möglich diese Daten zu bekommen (get Funktion und print Funktion) :

```
int getm() { return this->m; } // Anzahl der Zeile
int getn() { return this->n; } // Anzahl der Spalte
void print() {
```

Zweite Klasse ist eine „Gleichungssystem Klasse“, also ein Gleichungssystem kann lösen. In diesem Fall müssen wir auch die Matrix in einer Datei eingeben, aber hier müssen wir darauf achten, dass in der Matrix wird b Vektor eingebettet, deshalb hat die Matrix +1 Spalte. (in der Datei müssen wir die Anzahl der Spalte und Unbekannte eingeben, also dieses Vektor nicht):

$$x_1 \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix} + x_2 \begin{pmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{pmatrix} + \cdots + x_n \begin{pmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Der Benutzer müssen nur eine Funktion rufen (losen()) und das Programm wird die Lösungen ausschreiben (oder wenn es nicht lösbar, das Problem). Im Hintergrund wird mehr Funktion rufen. Beim Gleichungssystem gibt es mehr Möglichkeit, wie die Matrix „aussehen“ kann, also es gibt mehr Unbekannten als Gleichungen oder umgekehrt. Im Programm gibt es 2 Funktion, die die größe Menge der Lösungen durchführen. Der Name diesen Funktionen ist gauss1/gauss2 (wegen der Gauss-Elimination). In der Hauptfunktion (losen()) wird gauss1 oder gauss2 rufen. Es hängt davon ab, wie viele Unbekannten und Gleichungen gibt. Wenn es mehr Unbekannten als Gleichungen gibt, wird gauss1 gerufen, andernfalls gauss2. Beide Funktionen beruhen auf Gauss-Elimination aber die Daten sind verschieden.
Während der Gauss-Elimination müssen wir darauf achten, dass das Programm nur richtige Operationen durchführen dürfen (mit 0 dividieren nicht usw.) Dieses Problem habe ich mit der Funktion prüfen() gelöst. Diese Funktion checkt, ob in die erstes Element 0 ist. Wenn ja, dann wird die 1. Zeile mit 2. Zeile oder mit 3.Zeile (wenn arr[1][0] auch 0) umtauschen. Natürlich muss das Programm während der Lösung immer darauf achten, also andere Zeile Umtauschen wird in gauss1 und gauss2 eingebettet. Diese Umformungen ergeben die gleichen Lösungen, während des Umtauschen verändert sich die Lösungen des Gleichungssystems nicht. Prüfen Funktion:

```
template<class type>
void gleichung<type>::prüfen() {
    if (this->arr[0][0] == 0) {
        int i;
        type tmp;
        for (i = 1; this->arr[i][0] == 0; i++);
        for (int j = 0; j <= this->n; j++) {
            tmp = this->arr[0][j];
            this->arr[0][j] = this->arr[i][j];
            this->arr[i][j] = tmp;
        }
    }
}
```

Bei dieser Klasse ist es empfehlenswert double zu benutzen, weil es während der Lösung viele Bruchzahl gibt und int wird es abrunden.

Die dritte Klasse ist ein Vektor Klasse. Diese Klasse vererbt zwei Klassen: abstrakt Matrix Klasse und abstrakt Vektor Klasse. In dieser Klasse können wir 3D Vektoren transformieren in dem Descartes-Koordinatensystem. In dieser Klasse befindet sich zwei Arrays mit der Länge 3 (von der Klasse abstrakt Vektor) und 3 Matrizen (1 von der Klasse Matrix). In Mathe wird diese Transformation mit den Matrizen durchgeführt. Für jede Operation gehört ein 3x3 Matrix und mit der Hilfe diesen Matrizen können wir die neuen Koordinaten bekommen. Funktionen sind: Rotieren um x,y,z Achse, Projektion auf xy,xz,zy Ebene, Spiegelung auf xy,xz,yz Ebene und Strecken/Stauchen die verschiedene Koordinaten. Ich habe diese Aufgabe in 3D verwirklicht, weil es in diesem Fall geometrisch vorstellbar ist. (und nützlich sein kann). Wenn wir 2 Operationen durchführen möchten, ist die Reihenfolge nicht egal! Diese Operationen verwendet Matrix Operationen. A*B! = B*A, wo A ist die Matrix von Rotieren, B ist die Matrix von Spiegelung zum Beispiel. In Aufgabe habe ich es so gelöst, dass die Reihenfolge von dem Benutzer bestimmt wird. Die verschiede Vektoren sind in verschiedenen Klasse gespeichert, also zu einer Klasse gehört pünktlich ein Vektor. Nach der Transformation wird der

```cpp
template <class type>
void Vector<type>::Rotxinit(double x) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            this->arr2[i][j] = 0;
            this->arr3[i][j] = 0;
        }
    }
    this->arr2[0][0] = 1;
    this->arr2[1][1] = cos(x*rad);
    this->arr2[1][2] = -sin(x*rad);
    this->arr2[2][1] = sin(x*rad);
    this->arr2[2][2] = cos(x*rad);
}
template <class type>
void Vector<type>::Rotx(double x) {
    this->Rotxinit(x);
    this->initarr1();
    this->lösen();
}
```

```cpp
template<class type>
void Vector<type>::lösen() {
    type tmp;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                this->arr3[i][j] += this->arr[i][k] * this->arr2[k][j];
            }
        }
    }
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            this->x[i] += this->v[j] * this->arr3[i][j];
        }
    }
    for (int i = 0; i < 3; i++) {
        tmp = this->v[i];
        this->v[i] = this->x[i];
        this->x[i] = tmp;
    }
}
```

originelle Vektor in einem anderen Vektor gespeichert.
In dem Programm wird die verschiedene Funktionen geschrieben, und für jede Funktion gehört eine „init“ Funktion, die die Matrizen initialisieren. Zum Beispiel:

In dieser Klasse ist es empfehlenswert double zu benutzen, weil es viele Bruchzahl zwischen 0 und 1 gibt(wegen sin, cos) und mit int werden diese Zahlen 0 sein.

Destruktor sieht so aus (arr sind vererbt, also dürfen wir es nicht befreien!(virtual)):

```cpp
virtual ~Vector() {
    for (int i = 0; i < 3; ++i) {
        delete[] this->arr2[i];
        delete[] this->arr3[i];
    }
    delete[] this->arr2;
    delete[] this->arr3;
}
```

Während der Hausaufgabe habe ich immer dynamische Memory Speicherung benutzt. Die Matrizen werden aus Dateien ausgelesen. Ich habe auch try-catch verwirklicht. Wenn wir etwas Fehler machen, wie zum Beispiel falsche Indexierung, falsche Datei oder etwas falsche Eingabe, das Programm wird nicht sofort terminieren, sondern es schreibt das Problem aus, und dann wird alle Memory befreien(automatisch) und nur danach wird es terminieren.