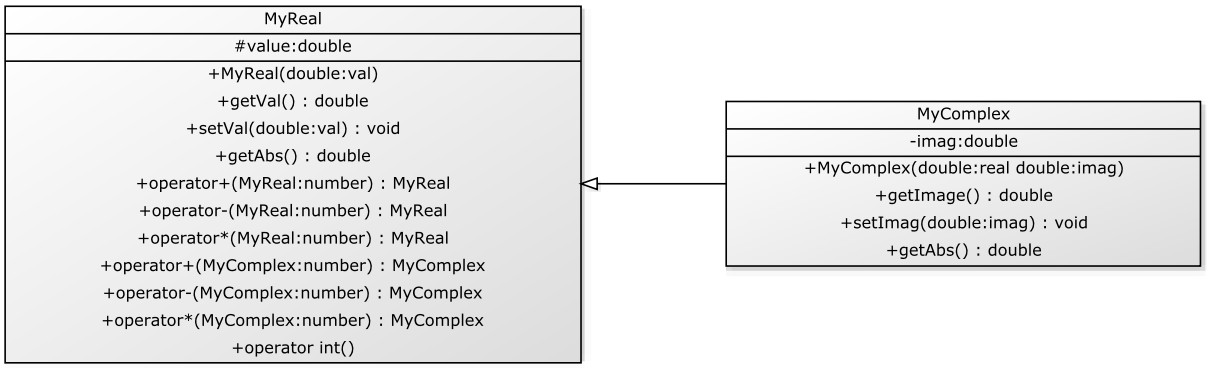
Dirk Praetorius,
Marcus Page

Wintersemester 2011
14. Dez. 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

In den Folgenden vier Aufgaben geht es darum Die Klassen `MyReal` und `MyComplex` mit Ihren Methoden zu implementieren wie sie im folgenden UML Diagramm angegeben sind.



Aufgabe 9.1*. *[Klassendefinition & Vererbung]* Schreiben Sie die Klassen `MyReal` und `MyComplex`, wobei `MyComplex` öffentlich von `MyReal` abgeleitet ist. Schreiben Sie zusätzlich `get-` und `set`-Methoden um auf die Felder `value` bzw. `imag` zugreifen zu können und Konstruktoren mit denen Sie Ihre Objekte initialisieren können. Speichern Sie den Source-Code unter `real_comp.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.2*. *[Methoden redefinieren]* Schreiben Sie eine Methode `getAbs()`, welche den Absolutbetrag der reellen Zahl zurückliefert. Redefinieren Sie die Methode in `MyComplex` so, dass hier der Betrag der komplexen Zahl zurückgegeben wird. Kennen Sie eine Möglichkeit die Methode `getAbs()` von `MyReal` auf eine komplexe Zahl anzuwenden um nur den Betrag des Realteils zu erhalten?

Aufgabe 9.3*. *[Operationen definieren & Typecasts]* Schreiben Sie sich Funktionen `add`, `multiply` und `subtract` mit denen Sie Werte vom Typ `MyReal` addieren, multiplizieren bzw. subtrahieren können. Implementieren Sie zusätzlich ein *Typecast* um Ihre reellen bzw. komplexen Zahlen nach `int` zu konvertieren (Es soll dabei nur der ganzzahlige Anteil des Realteils ausgegeben werden).

Aufgabe 9.4*. *[Operatoren definieren & überladen]* In der letzten Aufgabe haben Sie die Rechenoperationen noch mühsam mittels eigens geschriebenen Funktionen realisieren müssen. C++ erlaubt hierzu eine leichtere Syntax mittels Operatoren. Schreiben Sie für die obigen Rechenoperationen Operatoren, so dass Sie zwei Zahlen etwa mit der folgenden Syntax addieren können:

```
MyReal real1(121.5);
MyReal real2(17.83);
MyReal real3(0);
real3 = real1 + real2;
```

Überladen Sie die Operatoren zusätzlich so, dass auch gemischte Arithmetik möglich ist. Orientieren Sie sich hierbei an obigem Diagramm. Was sind die Vorteile von diesem Vorgehen?

Aufgabe 9.5. *[Klassendefinition & UML]* Konstruieren Sie eine Klasse `Polynomial` die ein Polynom p der Länge $n \in \mathbb{N}$ mittels seiner Koeffizienten $a_i \in \mathbb{R}$ abspeichert. Der Datentyp soll also

$$p = \sum_{j=0}^{n-1} a_j x^j \tag{1}$$

speichern können. Die Klasse soll außerdem die folgenden Funktionen beinhalten:

- `getZeros`: bestimmt die Nullstelle(n) des Polynoms (bis zum Grad 2) und gibt sie am Bildschirm aus. Hierbei soll zusätzlich überprüft werden ob der Grad des Polynoms kleiner als 3 ist und ansonsten eine entsprechende Meldung ausgegeben werden.
- `derivative`: bestimmt die erste Ableitung des Polynoms und gibt sie wieder als Polynom zurück.
- `antiderivative`: bestimmt eine Stammfunktion des Polynoms und gibt sie wieder als Polynom zurück.

Die Datenfelder sollen selbstverständlich als `private` deklariert werden, Sie müssen sich also Zugriffsfunktionen schreiben. Schreiben Sie außerdem Konstruktoren um das Nullpolynom, bzw. ein Polynom der Länge n zu initialisieren. Die Koeffizienten sollen hierbei im Konstruktor von der Tastatur eingelesen werden. Zusätzlich soll es die Möglichkeit geben zwei Polynome mittels `'+'` zu addieren. Erstellen Sie in dieser Aufgabe zunächst ein UML-Diagramm der Klasse `Polynomial` und schreiben Sie die Klassendefinition, Zugriffsfunktionen und Konstruktoren. Speichern Sie den Source-Code unter `Polynomial.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.6. *[Nullstellen & Operatoren]* Implementieren Sie die Funktion `getZeros`, sowie die Möglichkeit zwei Polynome mittels `'+'` zu addieren.
Tip: Es könnte sinnvoll sein einen neuen Konstruktor zu schreiben um das Zielpolynom leichter zu erzeugen.

Aufgabe 9.7. *[Polynome & Ableitungen]* Implementieren Sie die Funktionen `derivative` und `antidervative` wie oben beschrieben.

Aufgabe 9.8. *[Operatoren überladen]* Reelle Zahlen sind auch Polynome. Überladen Sie Ihre Additionsroutine so, dass sich auch Zahlen vom Typ `MyReal` zu Polynomen hinzuaddieren lassen. Implementieren Sie eine entsprechende Funktionalität auch in Ihre Klasse `MyReal`.

Aufgabe 9.9. *[Typecast]* Schreiben Sie einen Konstruktor für Ihre `Polynomial` Klasse, der es Ihnen erlaubt, per Zuweisung einer Zahl vom Typ `MyReal` direkt konstante Polynome zu erzeugen. Es soll also der folgende Code möglich sein:

```
MyReal real1(17.2);
Polynomial poly = real1;
```

Nun können Sie Polynome auf verschiedene Arten mit reellen Zahlen addieren. Vergleichen Sie diese Herangehensweise mit der vorherigen Aufgabe. Was sind die Vor- und Nachteile?

Aufgabe 9.10. *[Interpolation]* Schreiben Sie eine Funktion `interp` der vier Zahlen x_1, x_2, y_1, y_2 vom Typ `MyReal` übergeben werden und die die Punkte $(x_1, y_1), (x_2, y_2)$ linear interpoliert. Das Ergebnis ist ein Polynom und soll auch als solches zurückgegeben werden.