Draft der C++-Vorlesung vom 13.Mai 2022

Templates §10

Templatefunktion §10.1

Kurze Erinnerung an Templatefunktion von letzter Woche.
Umwandlung der Funktionen in *v_ 3a/* (verbesserte Geheimzahl) in Templatefunktionen.

Templateklasse §10.2

Analog zu Templatefunktionen wird eine Familie von Klassen erzeugt.

```
template<class T>
class X
{
...                          // Definition der Klasse X<T>
};
```

Erst mit der Belegung des Templateparameters wird eine Klasse erzeugt und ein Objekt dieser Klasse deklariert.

```
{
   X<int>    ai;
   X<float> fi;
   vector< X<char> > vc;
};
```

Umformulierung von `Komplex` aus *v_ 6a/* in eine Template-Klasse `Komplex<T>`: *v_ 8c/*

1. `template <class T>` vor die Klassendeklaration schreiben. (**.h*)
2. Den zu verallgemeinernden Datentyp durch den Template-Parmeter `T` ersetzen. (**.h, *.tcc*)
   Achtung, vielleicht wollen Sie nicht an allen Stellen diesen Datentyp ersetzen.
3. Ersetze in allen Parameterlisten, Returntypen und Variablendeklarationen den Klassentyp `Komplex` durch die Template-Klasse `Komplex<T>`. (**.h, *.tcc*)
4. Vor jede Methodenimplementierung `template <class T>` schreiben. (**.tpp*)
   Desgleichen vor Funktionen, welche die Klasse `Komplex::` als Parameter benutzen. (**.tpp*)
   Bei `friend`-Funktionen muß man `template <class T>` auch im Deklarationsteil vor der Funktion angeben. (**.h*)
5. In der Methodenimplementierung `Komplex::` durch `Komplex<T>::` ersetzen. (**.tpp*)
   Achtung: `Komplex::Komplex(...)` $\longrightarrow$ `Komplex<T>::Komplex(...)` , dasselbe beim Destruktor.
6. Im Headerfile das Sourcefile includieren, also `#include "komplex.tcc"`, oder gleich alles in das Headerfile schreiben (nicht empfohlen). (**.h*)

Weitere Topics:

- Mehrere Template-Parameter [Haase22, §10.2.2].

- `friend`-Funktionen und Template-Klasse [Haase22, §10.2.4].

Einschränkungen an Template-Datentyp §10.3

- Überprüfung via Type Traits [Haase22, §10.3.1]. *v_ 8c_ cpp17/*
- C++20: *Concepts* für Typüberprüfung [Haase22, §10.3.2]. *v_ 8c_ cpp20/ komplex2.h*

STL §11

Standard **T**emplate **L**ibrary: Bsp. aus Listing 11.1 des Skriptes.

- Container
- Algorithmen
- Iteratoren
- Container und Algorithmen wissen nichts voneinander $\Longrightarrow$ Iteratoren als Bindeglied.

Literatur

[Haase22] Gundolf Haase: Einführung in die Programmierung mit C++ (2022), *www*[1].

[Stroustrup10] Bjarne Stroustrup: Einführung in die Programmierung mit C++. Pearson Studium, München (2010).

[1] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`