Templates (Schablonen)

Eine gute Einführung in dieses Thema zeigt das folgende kurze Programmierbeispiel:

1. Problemstellung:

   Für Ganzzahlen, Gleitpunktahlen und Buchstaben sollen die Funktionen Maxi, Maxd und Maxc entworfen werden, die jeweils die größere zahl bzw. den größeren Buchstaben im Alphabet eines Zahlen- bzw. Buchstabenpaars zurückgeben.

   Zunächst müssen drei Funktionen definiert werden, welche die jeweils größere Zahl bzw. den höheren Buchstaben zurückgeben. Dazu eignet sich der Bedingungsoperator „?:". Er verarbeitet als einziger Operator drei Operanden. Eine Anweisung der Form a?b: c bedeutet: Wenn a true, dann gilt b, ansonsten c.
   In der Funktion Maxi bedeutet das folgendes: Ist x größer als y? Wenn ja, dann x, sonst y.

   In der Hauptfunktion main werden Variablen der drei Typen int, double und char definiert und die Funktionen damit aufgerufen. Main dient hierbei nur als Testprogramm für die Funktionen Maxi, Maxd und Maxc.

   ohneTemplates.cpp

   Fasst man das Gemeinsame der Funktionen zusammen, könnte man hierfür folgende theoretische Funktion mit dem „Joker" xxx schreiben:

```
xxx Max(xxx x, xxx y)
{
     return x > y ? x : y;
}
```

   Diese neue Funktion würde wie eine Schablone über die drei konkreten Funktionen passen. Und damit sind wir auch schon mitten im Thema Schablonen. Ziel ist es, die drei Funktionen durch eine gemeinsame, für alle Datentypen passende, Funktion zu ersetzen
   Hierzu wird eine Funktion mit parametrisierten Datentypen geschrieben, d.h. Für de noch unbestimmten Datentyp wird ein „Joker" oder Platzhalter eingesetzt, der später vom Compiler durch den tatsächlich erforderlichen Datentyp ersetzt wird.

2. Erzeugen von Funktionstemplate

   Ein Funktionstemplate ist eine Funktion, die für abstrakte Datentypen als Generierungsparameter definiert ist. Bei jedem Funktionsaufruf wird anhand der aktuellen Argumente die zum

Datentyp passende Funktion generiert.

Die allgemeine Syntax zur Erzeugung einer Funktionsschablone lautet:

template <class Typbezeichner>
Funktionsdefinition

Übertragen auf die Funktion Max ergibt sich folgender Quellcode:

mitTemplates.cpp

3. Probleme bei Funktionstemplates

Und das funktioniert immer? Hierzu ein kleiner Exkurs zum Thema Homonymauflösung – der ein oder andere erinnert sich sicherlich an das Kinderspiel Teekesselchen. Unter homonymen Funktionen versteht man allgemein Funktionen mit gleichen Namen (siehe hierzu auch überladene Funktionen) Die Homonymauflösung beinhaltet dabei das schrittweise Suchen der am besten passenden Funktion. Die Suche erfolgt dabei durch den Compiler zur Laufzeit des Programms. Dieser Vorgang erfolgt nach folgenden Regeln:

1. Gibt es eine passende Funktion mit exakter Übereinstimmung?
2. Gibt es eine Funktion, bei der die Datentypen der Argumente mit denen der formalen Parameter übereinstimmen?
3. Wenn nein, mit einer der folgenden Konvertierungen versuchen:
   a) Übereinstimmung durch integrale Promotion erzielen (z.B. char zu int oder auch float zu double)
   b) Standardkonvertierungen zulassen, d.h. In den jeweils „mächtigeren" Datentyp konvertieren. (z.B. unsigned int zu int)
   c) benutzerdefinierte Umwandlungen zulassen (z.B. durch Umwandlung mittels Konstruktoren)
   d) variable Parameterlisten zulassen

4. Welche Vorteile bieten Templates=

Neben dem Vorteil, Funktionen für verschiedene Datentypen bereitzustellen haben Templates in C++ auch weitere Vorzüge: Das Sprachelement template unter C++ enthält die für die C++ charakteristische Typprüfung. Der Compiler kann also zum Aufspüren von Programmfehlern genutzt werden. Die Fehleranfälligkeit ist geringer! C++ bietet mit der Standard Template Library (STL) eine standartisierte Bibliothek mit nützlichen Klassen und Funktionen, die mit den

unterschiedlichsten Typen zusammenarbeiten.