1 Templates

Templates (Schablonen) ermöglichen eine generische Programmierung mit Hilfe der Sprache C++.
In einem Template wird ein Algorithmus allgemein ohne einen Bezug auf Datentypen beschrieben. Die beschriebenen Aktionen können auf unterschiedliche Datentypen angewandt werden. Zum Beispiel eine Funktion, die das Minimum oder Maximum von einer bestimmten Anzahl von Werten zurückgibt, sucht unabhängig vom Datentyp immer auf dieselbe Weise den kleinsten oder größten Wert.

1.1 Funktions-Templates

Funktions-Templates dienen als Vorlage für die Generierung von gleichartigen Funktionen, die sich in den folgenden Punkten unterscheiden können:

- Dem Rückgabewert der Funktion.
- Dem Datentyp der Argumente, die an die Funktion übergeben werden.
- Dem Datentyp der lokalen Variablen.

Alle generierten Funktionen besitzen aber die gleiche Anzahl von Argumenten, die übergeben werden und enthalten die gleichen Anweisungen.

Beispiel:

```
short Min(short wert01, short wert02){
      short min;

      min = (wert01 < wert02? wert01 : wert02);
      return min;
}
long Min(long wert01, long wert02){
      long min;

      min = (wert01 < wert02? wert01 : wert02);
      return min;
}

double Min(double wert01, double wert02){
      double min;

      min = (wert01 < wert02? wert01 : wert02);
      return min;
}
```

Alle drei Funktionen bekommen zwei Argumente übergeben und berechnen den minimalen Wert. Die Funktionen unterscheiden sich nur in den Datentypen der Übergabeparameter, des Rückgabewertes sowie der lokalen Variablen.
Die Min-Funktionen werden jetzt zu einem Funktions-Template zusammengefasst.

```
template <typename TDATA>
TDATA Min(TDATA wert01, TDATA wert02){
      TDATA min;

      min = (wert01 < wert02? wert01 : wert02);
      return min;
}
```

In der Funktion werden alle Datentypen durch einen beliebigen Namen ersetzt. Mit Hilfe der Anweisung `template <typename TDATA>` wird dem Compiler mitgeteilt, dass der Datentyp später festgelegt wird.

Das Funktions-Template wird genauso wie eine Funktion aufgerufen.

```
int main(){
      const int wert = 3;
      const int zahl = 5;
      int min;

      min = Min(wert, zahl);
      std::cout << "Min: " << min;
}
```

Beim Übersetzen des Programms durch einen Compiler, wird zuerst überprüft, ob es eine Funktion passend zum Datentyp des Aufrufes gibt. Wenn ja, wird diese Funktion aufgerufen. Andernfalls wird nach einem Funktions-Template gesucht.
Ist ein passendes Funktions-Template vorhanden, wird der formale Datentyp durch den tatsächlichen ersetzt und eine Funktion automatisch generiert. Bei der Generierung der Funktion findet keine automatische Typkonvertierung statt.
Zum Beispiel: Einem Template werden Zeiger als Argumente übergeben. Bei einem Zeigervergleich werden die darin enthaltenen Adressen verglichen. D. h. die Adressen werden nicht durch die tatsächlichen Werte an den Adressen ersetzt.
Um solche Fehler zu umgehen, wird entweder Funktion für diese speziellen Datentypen oder ein spezialisiertes Funktions-Template geschrieben.

Beispiel für eine spezialisierte Funktion, die das kleinste char-Zeichen herausfiltert:

```
char* Min(char* ptr01, char* ptr02){
      if(strcmp(ptr01, ptr02) > 0){
            return ptr01;
      }
      else {
            return ptr02;
      }
}
```

Beispiel für ein Template, welches auf den char-Datentyp spezialisiert ist und das kleinste Zeichen herausfiltert.

```
template<>
char* Min<char*>(char* ptr01, char* ptr02){
      if(strcmp(ptr01, ptr02) > 0){
            return ptr01;
      }
      else {
            return ptr02;
      }
}
```

Die `template`-Anweisung besitzt leere spitze Klammern. Der gewünschte Datentyp wird nach dem Funktionsnamen in spitzen Klammern angegeben. Der Datentyp des Rückgabewertes wird explizit vor dem Funktionsnamen angegeben.

Ein Funktions-Template kann auch mehrere formale Parameter besitzen.

```
template <typename TDATA, typename TDATEN>
TDATEN Min(TDATA wert01, TDATEN wert02){
      TDATEN min;

      min = (wert01 < wert02? wert01 : wert02);
      return min;
}
```

Der Aufruf sieht folgendermaßen aus:

```
int main(){
      const int wert = 4;
      const float zahl = 3.99;
      float min;

      min = Min(wert, zahl);

      std::cout << "Min: " << min << '\n';
}
```

Der Compiler generiert beim Aufruf aus dem Template folgende Funktion:

```
float Min(int wert01, float wert02){
      float min;

      min = (wert01 < wert02? wert01 : wert02);
      return min;
}
```

Weitere Informationen können Sie in dem Buch

David Vandervoorde, Nicolai M. Josuttis
C++ Templates: The Complete Guide
Addison-Wesley 2002
ISBN 0201734842

nachlesen.