OOP – Objektorientierte Programmierung

Überblick über C++

Wilhelm Brezovits

1. Allgemeines

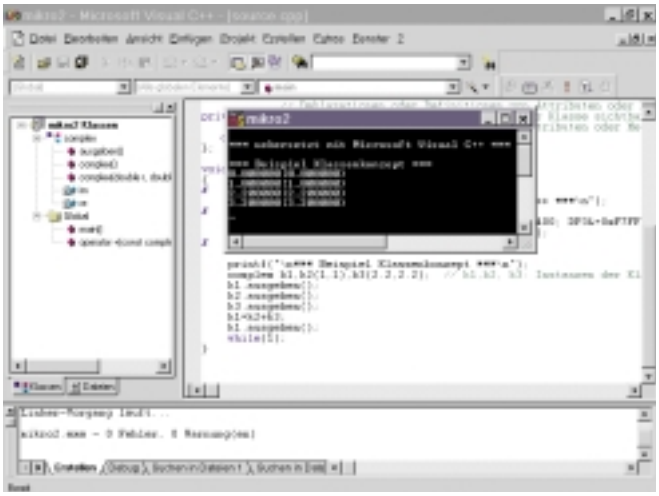
War vor Jahren bei der Programmierung von Mikrocontrollern noch die übliche Diskussion Assembler oder C, so ist heute das Diskussionsthema C, C++ oder EC++.

Embedded C++ (EC++) ist eine „Untermenge“ von C++. Durch „Weglassen“ von Teilen des C++ Sprachumfanges möchte man bei EC++ Laufzeit gewinnen. Eine Spezifikation von EC++ findet man unter *http://www.caravan.net/ec2plus/*.

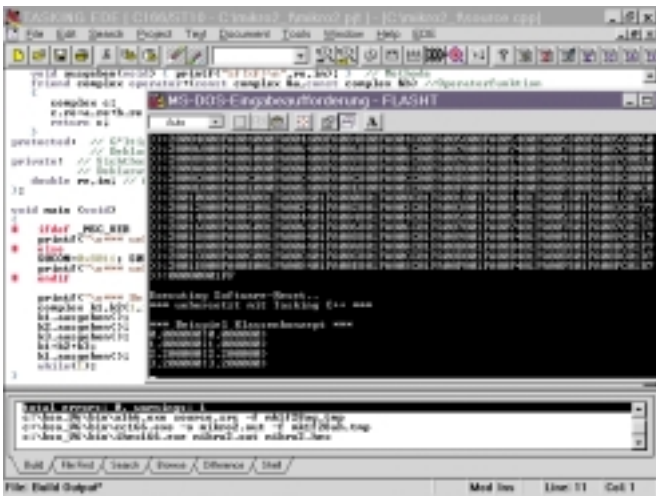
Die folgenden Programmbeispiele (gleicher Source Code!) wurden mit dem Microsoft Visual C++ Compiler und dem Tasking C++ Compiler übersetzt und erfolgreich auf verschiedenen Zielsystemen zum Ablauf gebracht.

C++ Compiler	Zielsystem
Microsoft Visual C++ (Version 6.0 Service Pack 2)	Intel-Mikroprozessor im PC
Tasking C++ (Version 6.0r3)	16-Bit-Mikrocontroller

Screenshot Microsoft Entwicklungsumgebung und „Ergebnis“ Beispielprogramm



Screenshot Tasking Entwicklungsumgebung und „Ergebnis“ Beispielprogramm



Im Gegensatz zum zentralen Aufbau eines Programms (ausgehend von `main()`) bietet die OOP die Möglichkeit, einzelne eigenständige Intelligenzinseln (Klassen) zu definieren.

Der größte Vorteil der OO-Programmierung ist die Wiederverwendbarkeit, d.h. Klassen müssen so entwickelt werden, dass sie alle Verantwortlichkeiten über ihre Daten und ihr Verhalten übernehmen und dadurch auch unter völlig verschiedenen Umgebungen in unterschiedlichen Situationen eingesetzt werden können.

Man unterscheidet zwischen zwei Arten der Wiederverwendung von Klassen: Benutzungsbeziehung („hat ein“) und Vererbungsbeziehung („ist ein“).

2. OOP im Detail

Die drei großen Säulen der objektorientierten Programmierung:

- Klassenkonzept (Kapselung durch Klassen)
- Vererbung (hierarchische Wiederverwendung in der Vererbung)
- Polymorphie (Vielgestaltigkeit)

2.1 Klassenkonzept

Im Prinzip ist eine Klasse (`class`) ein Datentyp, also eine komplexe Datenstruktur so wie in C ein Aufzähltyp (`enum`), eine Struktur (`struct`), ein Bitfeld oder eine Variante (`union`).

Klassen (Datenkapsel, Struktur) können als eine Erweiterung von Strukturen betrachtet werden und beinhalten Daten (Attribute, Eigenschaften, data member) und Funktionen (Methoden, Elementfunktionen, Member-Functions, Botschaften) in einer abgeschlossene Einheit.

Die Klasse muss die volle Verantwortung für ihre Daten und ihr Verhalten übernehmen!

Interne Daten und Funktionen werden innerhalb der Klasse versteckt (Zugriffsattribut, Gültigkeitsbereich: `private` oder `protected`).

Dem Anwender der Klasse wird eine Schnittstelle (`public`) zur Benutzung zur Verfügung gestellt.

Von einer Klasse können beliebig viele Variablen (Instanzen, Instanzvariablen, Objekte) erzeugt werden.

Beispiel

```
#include <stdio.h>
#ifdef _C166  // Compiler=Tasking C++
#include <reg167.h>
#endif

class complex  // complex = Name der Klasse
{
public: // Sichtbarkeitsbereich:
        // Schnittstelle der Klasse nach außen sichtbar
        // Deklarationen oder Definitionen
        // von Attributen oder Methoden:
   complex() : re(0),im(0) {}  // Konstruktor
   complex(double r,double i) : re(r),im(i) {} // Konstruktor
   void ausgeben(void) { printf("%f|%f)\n",re,im); }  // Methode
   friend complex operator+(const complex &a,const complex &b)
      //Operatorfunktion
   {
      complex c;
      c.re=a.re+b.re; c.im=a.im+b.im;
```

MTM "Tasking"

```
        return c;
     }
protected:// Gültigkeitsbereich:
          // in eigener und abgeleiteter Klasse sichtbar
          // Deklarationen oder Definitionen
          // von Attributen oder Methoden
private:  // Sichtbarkeitsbereich: nur innerhalb der Klasse sichtbar
          // Deklarationen oder Definitionen
          // von Attributen oder Methoden:
   double re,im; // Attribute
};

void main (void)
{
#ifdef  MSC VER     // Compiler=Microsoft C++
    printf("\n*** uebersetzt mit Microsoft Visual C++ ***\n");
#else               // Compiler=Tasking C++
    SOCON=0x8011; S0BG=0x0040;
    P3|=0x0400; DP3|=0x0400; DP3&=0xF7FF; S0TIR=1;
    printf("\n*** uebersetzt mit Tasking C++ ***\n");
#endif

    printf("\n*** Beispiel Klassenkonzept ***\n");
    // k1,k2, k3: Instanzen der Klasse
    complex k1,k2(1,1),k3(2.2,2.2);
    k1.ausgeben();
    k2.ausgeben();
    k3.ausgeben();
    k1=k2+k3;
    k1.ausgeben();
    while(1);
}
```

Konstruktoren und Destruktor

Zwei besondere Methoden (Konstruktor, Destruktor) möchte ich näher beschreiben.

Der Konstruktor hat den gleichen Namen wie die Klasse und wird beim Instanzieren (Erzeugen) eines Objekts der Klasse aufgerufen. Der Destruktor hat auch den gleichen Namen wie die Klasse, beginnt allerdings mit `~` und wird beim Zerstören der Instanz aufgerufen. Dies ist bei dynamisch angeforderten Speicher sehr praktisch, da der Destruktor diesen freigibt.

Operator Overloading

Definiert man z.B. eine Klasse für komplexe Zahlen und addiert man zwei Instanzen der Klasse, so muss die richtige Vorgehensweise bei der Addition definiert werden. Dazu schreibt man eine sogenannte Operator-Funktion. Der Name der Methode lautet `operator+`. Dadurch ist die Anweisung `k2+k3` in unserem Beispiel erst möglich.

2.2 Vererbung

Eine Klasse (abgeleitete Klasse) kann eine andere Klasse (Basisklasse) erben und damit existierenden Code wiederverwenden.

Natürlich kann eine Klasse auch viele andere Klassen erben (=Mehrfachvererbung).

In der abgeleiteten Klasse sind dabei alle Daten/Funktionen der Basisklasse(n) verfügbar.

Beispiel

```
#include <stdio.h>
#ifdef _C166
#include <reg167.h>
#endif

struct datum
{
  int tag,monat,jahr;
};

class mensch                    // Basisklasse
{
 protected:
   datum geburtstag;
   int groesse;
};

class mann : public mensch      // Vererbung,
                                // mann wird von mensch abgeleitet
{
 protected:
   char bart;
};

class frau : public mensch      // Vererbung,
                                // frau wird von mensch abgeleitet
{
 protected:
   int kinder;
};

class paar : public mann, public frau  // Mehrfachvererbung
{
   datum ehe;
};

void main (void)
{
#ifdef _MSC_VER
    printf("\n*** uebersetzt mit Microsoft Visual C++ ***\n");
#else
    SOCON=0x8011; S0BG=0x0040;
    P3|=0x0400; DP3|=0x0400; DP3&=0xF7FF; S0TIR=1;
    printf("\n*** uebersetzt mit Tasking C++ ***\n");
#endif

    printf("\n*** Beispiel Vererbung ***\n");
    paar paar1, paar2, paar3;  // 3 Instanzen der Klasse paar
                               // werden erzeugt
    while(1);
}
```

2.3 Polymorphie

Polymorphie beschreibt die Fähigkeit, an unterschiedliche Objekte (entstanden durch Vererbung, also Ableitung) gleichnamige Botschaften zu senden, die abhängig vom Objekt zu unterschiedlichen Reaktionen führen.

Das bedeutet, bei einem Methodenaufruf wird anhand des Typs des gebundenen Objektes vom Compiler realisiert durch eine vom Compiler selbst generierte VMT-Tabelle (virtuelle Methoden Tabelle) herausgefunden, welche Methode er rufen soll.

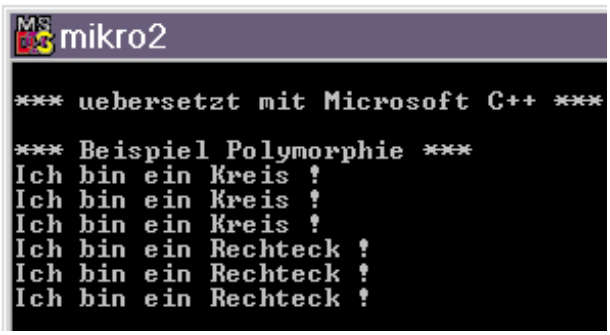
Beispiel

```
#include <stdio.h>
#ifdef _C166
#include <reg167.h>
#endif

class Zeichnung                          // abstrakte Basisklasse
{
public:
   virtual void drucke() = 0; // pure virtuelle Funktion
   // aufgrund "= 0" kann keine Instanz von Zeichnung angelegt werden
   // aufgrund des Steuerwortes virtual generiert der Compiler
   // eine VMT-Tabelle
};

class Kreis : public Zeichnung           // abgeleitete Klasse
{
public:
   void drucke()
   {
      printf("Ich bin ein Kreis !\n");
   }
};

class Rechteck : public Zeichnung        // abgeleitete Klasse
{
public:
   void drucke()
   {
      printf("Ich bin ein Rechteck !\n");
   }
};

void main (void)
{
#ifdef  MSC VER
    printf("\n*** uebersetzt mit Microsoft Visual C++ ***\n");
#else
    SOCON=0x8011; S0BG=0x0040;
    P3|=0x0400; DP3|=0x0400; DP3&=0xF7FF;  S0TIR=1;
    printf("\n*** uebersetzt mit Tasking C++ ***\n");
#endif
```

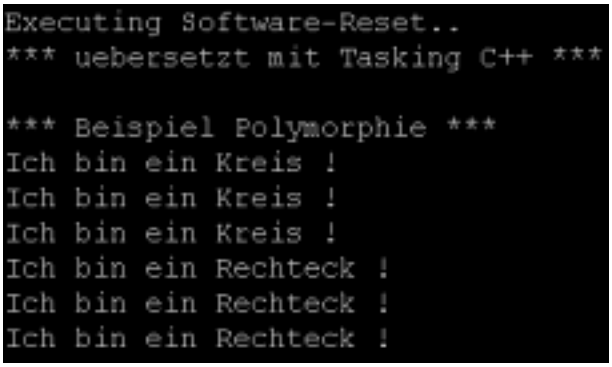
```
    printf("\n*** Beispiel Polymorphie ***\n");

    Kreis kreis1;           // 1 Objekt der Klasse Kreis anlegen
    Rechteck rechteck1;     // 1 Objekt der Klasse Rechteck anlegen

    Zeichnung *p;           // Basisklassenzeiger anlegen

    kreis1.drucke();        // "statische Variante" der Polymorphie
    p=&kreis1;
    p->drucke();            // dynamische Variante der Polymorphie
    (*p).drucke();          // andere Schreibweise

    rechteck1.drucke();     // "statische Variante" der Polymorphie
    p=&rechteck1;
    p->drucke();            // dynamische Variante der Polymorphie
    (*p).drucke();          // andere Schreibweise

    while(1);
}
```

Ergebnis

Screenshot „Ergebnis“ Microsoft C++ Compiler



Screenshot „Ergebnis“ Tasking C++ Compiler



3. Templates (Codeschablonen)

Natürlich beherrschen Mikrocontroller-C++-Compiler auch Templates:

Bei den Grundalgorithmen der Informatik (Listen, Bäume, u.s.w.) ist der Algorithmus immer der gleiche, nur der Datentyp der verwalteten Elemente ist immer ein anderer.

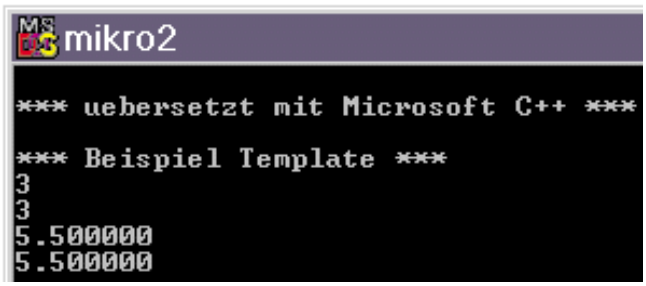
Templates bieten die Möglichkeit, einen oder mehrere Platzhalter für Datentypen bei der Definition von Funktionen oder Klassen anzugeben und so einen allgemeingültigen, typunabhängigen aber typsicheren Algorithmus zu formulieren, wobei der Compiler die entsprechende Funktion/Klasse erst bei der Verwendung für die Platzhalter generiert.

Beispiel

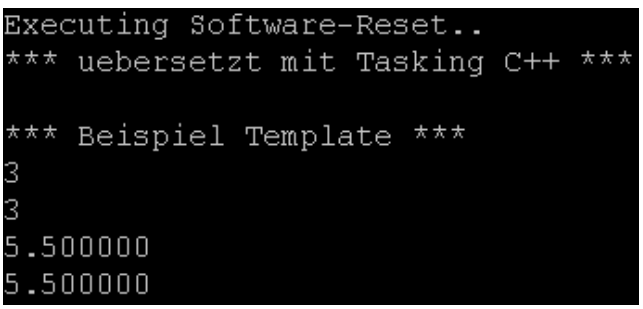
```
#include <stdio.h>
#ifndef  MSC VER
#include <reg167.h>
#endif

// Achtung:
// Typen wie int oder double koennen als bereits definierte Klassen
// betrachtet werden, der Operator > ist hier also bereits definiert.
// Bei selbstdefinierten Typen (Klassen) müssen alle vorkommenden
// Operationen (Operatoren) selbst definiert werden
// (Operator-Overloading).

template <class T1>    // T1... Platzhalter für Template
T1 max(T1 a, T1 b)
{
    return a>b?a:b;
}

void main(void)
{
#ifdef  MSC VER
    printf("\n*** uebersetzt mit Microsoft Visual C++ ***\n");
#else
    S0CON=0x8011; S0BG=0x0040;
    P3|=0x0400; DP3|=0x0400; DP3&=0xF7FF; S0TIR=1;
    printf("\n*** uebersetzt mit Tasking C++ ***\n");
#endif

    printf("\n*** Beispiel Template ***\n");

    int a=2,b=3;
    double x=5.5,y=4.4;

    printf("%d\n",max(a,b));    // entspricht: max(int,int)
    printf("%d\n",max(b,a));    // entspricht: max(int,int)

    printf("%f\n",max(x,y));    // entspricht: max(double,double)
    printf("%f\n",max(y,x));    // entspricht: max(double,double)

    while(1);
}
```

Ergebnis

Screenshot „Ergebnis“ Microsoft C++ Compiler



Screenshot „Ergebnis“ Tasking C++ Compiler



Schlussbemerkung

Für weitere Informationen möchte ich die WWW-Seite eines C, C++ und EC++ Compiler-Herstellers für die Infineon 16-Bit- und 32-Bit-Mikrocontrollerfamilien anführen:

http://www.tasking.com/products/80C166/

Ich hoffe, der Artikel konnte dem Leser einen Überblick über C++ bieten.

Vielleicht animiert er sogar, Software für die Infineon 16-Bit- oder 32-Bit-Mikrocontroller ab sofort in C++ zu implementieren?