2 Geschichte und Eigenschaften

In diesem Kapitel werden wir kurz auf die Geschichte von C++ eingehen. Es handelt sich dabei um die wohl einflussreichste Entwicklung der Programmiersprachen der letzten 30 Jahre[MIT10]. Wir werden durch ein C-Programm einen Übergang von C nach C++ skizzieren. Wenn Sie keine C-Erfahrung haben, so müssen Sie einfach gewisse Dinge so akzeptieren, wie sie dargestellt sind. Es wäre allerdings günstig, wenn Ihnen der Begriff der Struktur (struct), der bei diesem Übergang fundamental ist, bekannt ist. Anderenfalls verweisen wir auf die Folgekapitel oder Sekundärliteratur[Dus11].

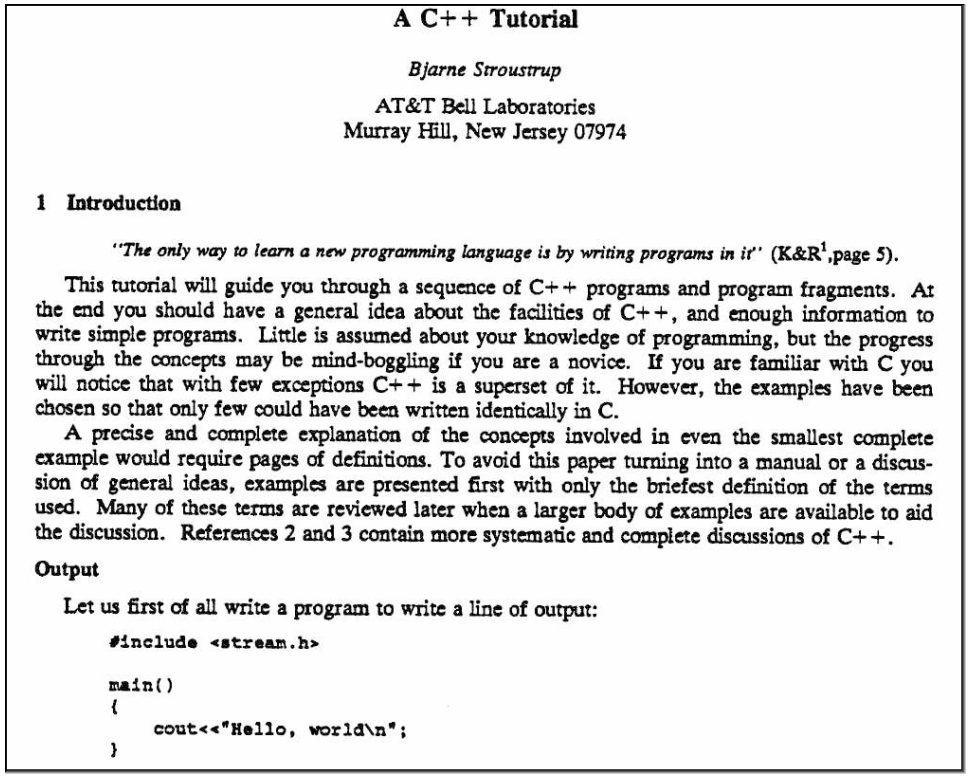
2.1 Geschichte der Sprache C++

Die Sprache C++ wurde Ende der siebziger/Anfang der achtziger Jahre von Bjarne Stroustrup[1] bei AT&T als eine Erweiterung von C entwickelt. Ziel war, größere Programme wartbarer zu machen, die Anzahl von Software-Fehlern zu verringern und Code-Redundanzen zu vermeiden. Natürlich gab es ständig Ideen, die in diese Richtung gegangen sind. Die Wirth´sche Sprachfamilie ist ein ebensolcher Ansatz. Hier wollen wir besonders die Sprache Modula 2 hervorheben. Gelang es doch Niklaus Wirth[2] mit dem Modulkonzept einen entscheidenden Schritt in Richtung OOP zu gehen. Man konnte Strukturen, die hier RECORD hießen, definieren und mit den Schlüsselwörtern
EXPORT QUALIFIED
gelang es, Funktionen oder Variablen ”versteckt” zu halten oder auch nicht. Seinen objektorientierten Niederschlag fanden diese Sprachen (Pascal, Modula) in Oberon und Delphi. Doch kommen wir zurück zu C++: 1979 entwickelte Stroustrup C mit Klassen. Unter einer Klasse, die Stroustrup class nannte, verstehen wir eine Struktur (struct) mit besonderen Eigenschaften. Diese Eigenschaften werden wir noch kennenlernen. Damit sich diese besondere Struktur von den in C üblichen Strukturen abhebt, war es sinnig einen neuen Namen zu definieren, der dann class war. Im Jahre 1983 entwickelt Stroustrup dann C++. In diesem Jahr wurde auch der Name der Sprache festgelegt: C++. Als wichtige Features entstanden virtuelle Funktionen und Operator Overloading. 1985 publizierte Stroustrup C++ als Erweiterung zu C (Release 1.0); dieses Jahr gilt als das Jahr der allgemeinen Verfügbarkeit der Sprache C++. Allerdings fehlte immer noch ein Compiler. Im Jahr 1988 erschien eine Art C++ Tutorial von Stroustrup. In diesem Dokument wurde das berühmte ”Hello world” auf C++ codiert; siehe dazu Abbildung 2.1. Im gleichen Jahr stand auch der erste kommerzielle Compiler zur Verfügung[att89]; siehe:
www.softwarepreservation.org/projects/c_plus_plus/cfront/release_2.0/doc/ReleaseNotes.pdf

[1] Bjarne Stroustrup (* 30. Dezember 1950 in Aarhus, Dänemark) ist Professor der Informatik an der Texas A&M University.

[2] Niklaus Wirth (* 15. Februar 1934 in Winterthur, Schweiz) ist ein Schweizer Informatiker.

D. Duschl, *Softwareentwicklung mit C++*, Studienbücher Informatik, DOI 10.1007/978-3-658-18123-9_2



A C++ Tutorial

Bjarne Stroustrup

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

1 Introduction

"The only way to learn a new programming language is by writing programs in it" (K&R[1],page 5).

This tutorial will guide you through a sequence of C++ programs and program fragments. At the end you should have a general idea about the facilities of C++, and enough information to write simple programs. Little is assumed about your knowledge of programming, but the progress through the concepts may be mind-boggling if you are a novice. If you are familiar with C you will notice that with few exceptions C++ is a superset of it. However, the examples have been chosen so that only few could have been written identically in C.

A precise and complete explanation of the concepts involved in even the smallest complete example would require pages of definitions. To avoid this paper turning into a manual or a discussion of general ideas, examples are presented first with only the briefest definition of the terms used. Many of these terms are reviewed later when a larger body of examples are available to aid the discussion. References 2 and 3 contain more systematic and complete discussions of C++.

Output

Let us first of all write a program to write a line of output:

```
#include <stream.h>

main()
{
    cout<<"Hello, world\n";
}
```

Abbildung 2.1: Der Anfang des Tutorials der Release 2.0 von B. Stroustrup.

Stroustrup verwendete einen C++-Translator, der seinen C++-Code in C-Code umsetzte und anschließend wurde dieser Code einem "normalen" C-Compiler übergeben:

C++ Code $\Rightarrow$ **Translator** $\Rightarrow$ **C Code** $\Rightarrow$ **C-Compiler** $\Rightarrow$ **Executable**

Im Release 2.0 wurde erstmals die Mehrfachvererbung definiert und es war möglich, weitere Operatoren zu überladen. Wir werden zu gegebener Zeit auf diese Features eingehen. So viel sei vorweg genommen: diese Features haben nicht nur Vorteile. 1991 wurden Templates implementiert und im Jahr 1998 wurde C++ erstmals als ISO/IEC 14882 standardisiert. Wir sollten an dieser Stelle erwähnen, dass der Begriff Template fundamentale Bedeutung in C++ hat. Sie sollten - wie schon beim Begriff der class an den Begriff struct - sich bei Template an das Feature Makro, das wir von der Sprache C her kennen, erinnern. Natürlich werden wir auch in diesem Buch nochmals darauf eingehen.

2.2 Eigenschaften

C++ ist eine multi-paradigmen Sprache. Das bedeutet, dass man sowohl prozedural, modular, objektorientiert als auch generisch (typunabhängig) programmieren kann:

- prozedurale Programmierung
  - Blockstrukturen
  - Funktionen
- modulare Programmierung
  - Namensräume
  - Überladen von Funktionen
  - Nutzung von ”static”
- objektorientierte Programmierung
  - Klassen
  - Vererbung
  - Polymorphie
- generische Programmierung
  - Templates (Funktion und Klassen)
  - Verwendung von auto-Variablen

Alle diese Eigenschaften werden wir im Verlauf des Buches detailliert kennenlernen und üben. Aus diesem Grunde soll hier nicht weiter darauf eingegangen werden.
Eine Bemerkungen sei jedoch noch erwähnt:
C++ ist wie schon C eine formatfreie Sprache; das heißt, Sie können Ihren Quellcode gestalten, wie Sie wollen. Wenn Sie jedoch eine gute Wiederverwendbarkeit Ihres Codes im Fokus haben, so sollten Sie unbedingt auf eine gute Lesbarkeit Ihres Codes achten. Und wenn man an eine gute Lesbarkeit des Programmes denkt, so hat man zwei Dinge im Auge:
A) Das Programm sollte eine vernünftige Form haben.
B) Das Programm sollte gut dokumentiert sein.
Unter A) verstehen wir zunächst, dass die geschweiften Klammern ({ und }) immer symmetrisch untereinander stehen. Unter B) wollen wir zunächst nur die Dokumentation der Software innerhalb des Quellcodes verstehen. Der Quellcode kann durch Einzeilenkommentare mit // kommentiert werden[3]. Welche Grundregeln sollten Sie beim Kommentieren beachten:

- Es ist besser, zu viel als zu wenig zu kommentieren. Trotzdem sollte der Kommentar kurz und treffend sein!
- Es ist besser, nichts zu kommentieren als falsch zu kommentieren.
- Trivialen Quellcode zu kommentieren ist überflüssig und schädlich, weil so etwas das Programm künstlich aufbläht und damit unleserlich macht.
- Es ist günstig, ein gewisses Format beim Kommentieren einzuhalten. Das erhöht die Lesbarkeit.

Grundsätzlich kann man es nicht oft genug wiederholen: kommentieren Sie klug, treffend und sauber und pflegen Sie ihre Kommentare! Das wird Ihnen die Weiterentwicklung Ihrer Software und das Einarbeiten in Fremdsoftware, sofern diese gut kommentiert ist, enorm erleichtern und

[3] Bitte beachten Sie, dass Syntax nicht zu kommentieren ist. Damit würden Sie das Programm unleserlich machen. Man sollte ebenfalls vermeiden, trivialen Quellcode zu kommentieren.

Ihnen sehr, sehr viel Zeit sparen.

In vielen Firmen bemüht man sich in englischer Sprache zu kommentieren. Das ist in der Regel auch richtig. Aber: wir sollten nicht vergessen, dass es manchem Entwickler kaum gelingt, die Quintessenz seines Programmes, seiner Funktion, seines Moduls oder seiner Klasse in deutscher Sprache kurz, knapp und verständlich zu beschreiben. Wie soll er das in englischer Sprache tun? In englischer Sprache kann diese Beschreibung nur unverständlicher sein. Dennoch muss man sagen, dass wesentliche Begriffe aus dem Bereich der Informatik ihre Wurzeln im angelsächsischen Sprachbereich haben. Aus diesem Grunde sollten die Entwickler entsprechend geschult sein, Englisch verständlich dokumentieren zu können. Wenn diese Qualifikation fehlt, ist eine der Dokumentation in der Landessprache vorzuziehen.

Hinweis: *Mangelnde Kommentare und Dokumentationen sind erfahrungsgemäß eines der Hauptprobleme in der Softwareentwicklung und treiben die Kosten von Pflegeaufwänden und Neuentwicklungen enorm in die Höhe. Eine Software kann erst dann als "fertig" bezeichnet werden, wenn ihre Dokumentation sauber beendet ist. Und es ist sicherlich nicht ganz falsch anzunehmen, dass die Qualität des Codes des Entwicklers in der Regel den gleichen Level hat wie die Dokumentation. Insofern kann man in erster Näherung einen guten Entwickler auch an der Qualität seiner Dokumentation, an der Qualität seiner Code-Kommentare ausmachen.*

2.3 Von C nach C++

Bevor wir ein Beispiel zeigen, wie man von C nach C++ übergehen kann, wollen wir uns klarmachen, wie man überhaupt ein Programm erstellen kann. Bei der Erstellung eines Programmes (C oder C++) werden drei Abschnitte durchlaufen:

- Sie editieren ein (oder mehrere) Programmfile(s) mit dem Visual Studio.
- Sie compilieren das Programm (Taste F7 im Visual Studio; s. Tabelle 1.2 auf Seite 8).
- Sie linken das Programm (wird ebenfalls mit F7 durchgeführt).

In Abbildung 2.2 ist der prinzipielle Ablauf der Entstehung eines lauffähigen Programmes dargestellt. In der IDE [4] des Visual Studios wird auch ein Assemblerlisting erstellt. Dieses Listing kann erzeugt werden, wenn Sie im Debugger - Sie befinden sich im Quellcodefenster in einem breakpoint - die rechte Mouse-Taste drücken und die Funktion

```
''Gehe zu Disassembly''
```

aktivieren (siehe Abbildung 2.3 auf Seite 27). Dann erhalten Sie den in der Abbildung 2.4 auf Seite 28 dargestellten Assemblercode unseres Programmes P001. Auch diesen Code können Sie debuggen und es ist durchaus in gewissen - wenn auch zugegebenermaßen wenigen - Situationen günstig, sich des Assemblerlistings im Debug-Mode zu bedienen. Dieses Debuggen dient Ihnen manchmal dazu, Ihr Programm besser zu verstehen, wenn Sie mit dem Sourcecode-Debuggen am Ende Ihres Lateins sind. Wenn beispielsweise Ihre Software nicht korrekt gebaut wurde, hilft

[4] IDE steht für "integrated development environment".

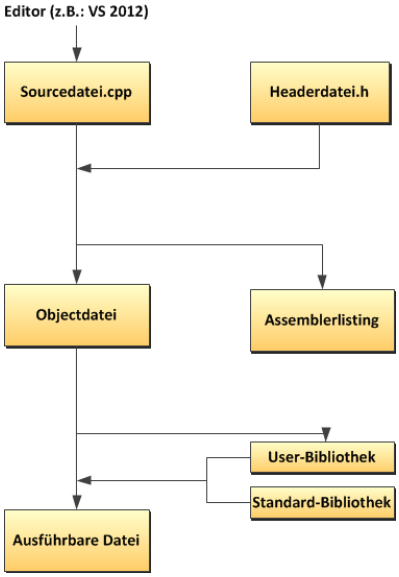


Abbildung 2.2: Ablauf der Entstehung eines Programmes.

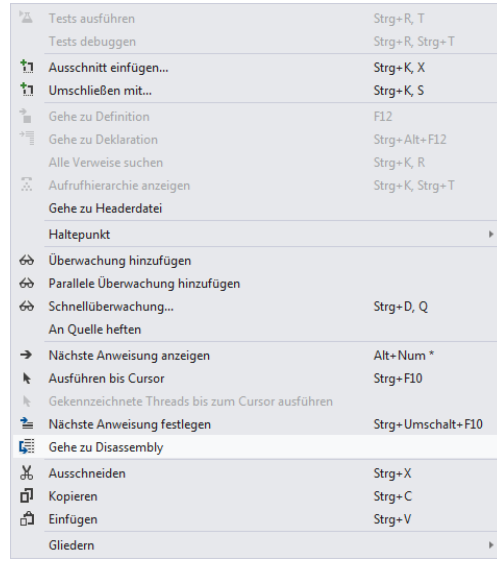


Abbildung 2.3: Aktivierung des Assemblerlistings der Source.

Ihnen das Sourcecode-Debuggen häufig nicht weiter. Wenn wir im Laufe des Buches die entsprechenden Debug-Fähigkeiten besitzen, können Sie einfach versuchen, ein geändertes Programm zu debuggen, ohne es zu compilieren und zu linken. Sie werden sehen, dass dies möglich ist und dass der Assemblercode Ihnen verrät, wo der Hase im Pfeffer liegt.

Mit dem Visual Studio 2012 können Sie auch C-Programme schreiben. Wie wir das tun können ist bereits im Buch [Dus11] beschrieben. Als Beispiel für ein Programm wählen wir das in unserem Vorwort skizzierte Beispiel einer Flüssigkeit. Wenn wir Flüssigkeiten beschreiben wollen, die die Eigenschaften haben, die auf Seite 28 beschrieben sind, so werden wir diese Eigenschaften in eine Struktur pressen. Sie sehen schon hier, dass Sie einigermaßen mit der C-Syntax vertraut sein müssen. Wenn Sie auf diesem Gebiet noch Probleme haben, so ist dies nicht weiter tragisch. Im Kapitel Grundlagen werden wir uns nochmals mit dieser Syntax beschäftigen. Wie dem auch sei: der Begriff der Struktur ist von fundamentaler Bedeutung. Das Schlüsselwort einer Struktur ist in C/C++

```
struct
```

Eine **Struktur** besteht aus Komponenten, die sich aus den Basistypen der Sprache C/C++ (wir sprechen von vier Basistypen: `char`, `int`, `float` und `double`) zusammensetzen. In einer Struktur können aber neben den Basistypen auch abgeleitete Typen (z.B. `unsigned short int`, `short`, ...) und Substrukturen deklariert sein. Strukturen fassen also mehrere primitive und/oder komplexe Variablen zu einer Einheit zusammen.

Diese Variablen in einer Einheit zusammenzufassen macht insofern Sinn, insofern man in dieser Einheit ein ”Problem” beschreiben kann; und dies mit nur einer Variablen, der sogenannten

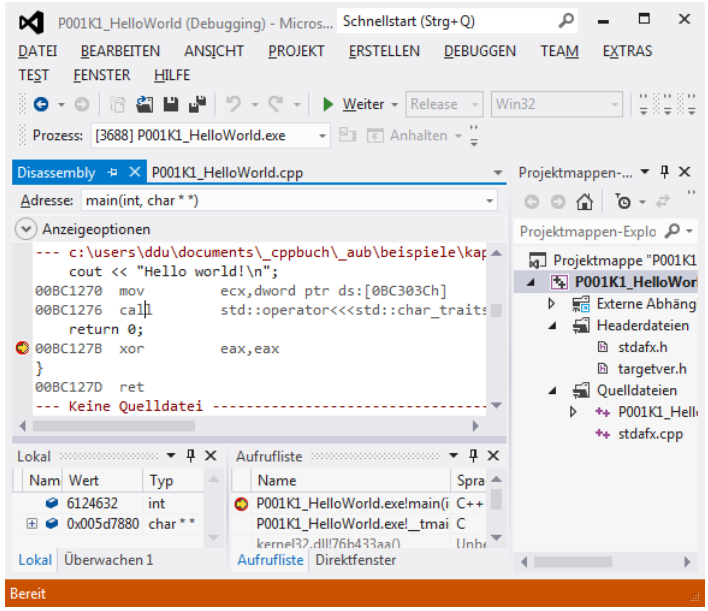


Abbildung 2.4: Source und Assemblerlisting im Visual Studio.

Strukturvariablen. Der Zugriff auf die Elemente der Struktur - auch Members oder Membervariablen genannt - ist via Punktoperator (.) oder Pfeiloperator (->) möglich. Sie können den Punktoperator-Zugriff im Programm-Beispiel 2.1 auf Seite 29 sehen.
Unsere Struktur hatte folgende Bestandteile:

- eine Farbe,
- eine chemische Formel,
- ein Volumen und
- eine Masse.

In C++ Code gegossen kann der Code unserer Struktur wie folgt ausschauen:

```
struct
{

   char strFarbe[32];
   char strFormel[32];
   float fVolumen;
   float fMasse;

};
```

Wir wollen dieses Kapitel nicht abschließen, ohne zu den oben erwähnten Bemerkungen ein Programm zu schreiben, das den Übergang von C nach C++ skizziert. Für dieses Programm ist der Begriff des Funktionspointers unverzichtbar. Wir werden ihn im Programm 2.1 anwenden und zugleich erläutern. Wenn Sie auf der Zeile `return 0` - also beim Kommentar //21 des Program-

```c
//P001K2_C2CPP.c
#include <stdio.h>                                                          //1
#define iStrLen_ 32                                                         //2

struct tLiquid                                                              //3
{
    char cFarbe[iStrLen_];                                                  //4
    char cFormel[iStrLen_];                                                 //5
    double dMasse;                                                          //6
    double dVolumen;                                                        //7
    double (*dPtr2RetDichte)(double, double);                               //8
};

double retDichte (double dMasse, double dVolumen)                           //9
{
    return dMasse/dVolumen;
}

int main (void)
{
    struct tLiquid liquid1;                                                 //10
    struct tLiquid liquid2;                                                 //11

    liquid1.dMasse    = 1;                                                  //12
    liquid1.dVolumen = 1;                                                   //13
    liquid1.dPtr2RetDichte  = retDichte;                                    //14
    strcpy_s (liquid1.cFarbe, iStrLen_, "transparant");                     //15
    strcpy_s (liquid1.cFormel, iStrLen_, "H2O");                            //16
    printf ("Es handelt sich um eine Fluessigkeit die %s ist und\n"
            "der chemischen Formel %s genuegt.\n",
            liquid1.cFarbe, liquid1.cFormel);                               //17
    printf ("Die Fluessigkeit hat eine Dichte von %.2f g/qcm.\n",
            liquid1.dPtr2RetDichte(liquid1.dMasse, liquid1.dVolumen));      //18
    printf ("Die Fluessigkeit heisst Wasser.\n");                           //19

    liquid2.dMasse    = 0.88;                                               //20 ff
    liquid2.dVolumen = 1;
    liquid2.dPtr2RetDichte  = retDichte;
    strcpy_s (liquid2.cFarbe, iStrLen_, "transparant");
    strcpy_s (liquid2.cFormel, iStrLen_, "C6H6");
    printf ("\nEs handelt sich um eine Fluessigkeit die %s ist und\n"
                        "der chemischen Formel %s genuegt.\n",
                        liquid2.cFarbe, liquid2.cFormel);
    printf ("Die Fluessigkeit hat eine Dichte von %.2f g/qcm.\n",
                        liquid2.dPtr2RetDichte(liquid2.dMasse, liquid2.dVolumen));
    printf ("Die Fluessigkeit heisst Benzol.\n");

    return 0;                                                               //21
}
```

Programm 2.1: P001K2_C2CPP mit Einzeilenkommentaren.

mes 2.1 - einen breakpoint mittels F9 setzen, dann erhalten Sie die Ausgabe wie in Abbildung 2.5 zu sehen.

//1 Nach dem Nummernzeichen # für den Präprozessor findet der Präprozessor die Anweisung ”include”. Er weiß damit, dass die Lib-Datei stdio.h an dieser Stelle in das Programm einzubauen ist. Das ist nötig, damit das Programm P001 kompilierbar ist. Wir verwenden nämlich die Funktion printf, die in stdio.h deklariert ist. Wenn die zu inkludierende Datei in spitzen Klammern steht, so sucht das System ”Visual Studio” das File zuerst in den

Systempfaden; falls das zu inkludierende File in Anführungszeichen steht, so sucht das System zuerst in den aktuellen Projektpfaden.

//2 Die Direktive #define ersetzt jedes weitere Vorkommen des Bezeichners iStrLen_ im Quelltext durch den String 32. Dies wird durch den Präprozessor erledigt. Wir haben es also hier mit einer Art SearchAndReplace zu tun.

//3 An dieser Stelle deklarieren wir eine Struktur. Mit dieser Struktur wollen wir Flüssigkeiten beschreiben.
Eine Struktur wird immer mit dem Schlüsselwort `struct` eingeleitet. Anschließend folgen:

- geschweifte Klammer auf ({)
- geschweifte Klammer zu (})
- Semikolon (;)

Es empfiehlt sich also immer, den hier skizzierten Rahmen zu codieren und anschließend die Variablen einzufügen.

//4 Nun beginnen wir die Variablen in der Struktur tLiquid zu deklarieren. Wir deklarieren Variablen, die nach unserer Ansicht die Flüssigkeit beschreiben sollen. Dies soll zunächst die Farbe der Flüssigkeit sein.
Bitte beachten Sie, dass die Typen immer mit einem Präfix eingeleitet werden. Dieser Präfix soll an den Typ erinnern: c für char, i für int, d für double. Wir werden an geeigneter Stelle nochmals auf eine sinnvolle Verwendung von Präfixen eingehen.

//5 An dieser Stelle wollen wir die chemische Formel der Flüssigkeit beschreiben,

//6 hier die Masse und

//7 hier das Volumen. Für Masse und Volumen verwenden wir als Variablentyp double.

//8 Final deklarieren wir einen Funktionspointer. Wie viele Sprachen besitzt auch die Sprache C die Eigenschaft, dass man nicht nur bei Variablen einen Pointer deklarieren kann, sondern auch bei Funktionen. Hier deklarieren wir eine Funktion, die aus den Membervariablen Masse und Volumen (dMasse, dVolumen) die Dichte der Flüssigkeit berechnet. Es ist einfach sinnvoll, in der Struktur auch die Funktion der Dichteberechnung zu deklarieren; wir haben ja alle Daten zusammen - warum sollte die Funktion fehlen? Sie gehört einfach dazu! In diesem Kontext sprechen wir auch von Memberfunktionen.
Unserer Memberfunktion werden zwei double-Variablen übergeben; Masse und Volumen. Und wir erhalten eine double Variable zurück - die Dichte der Flüssigkeit.

//9 Hier wird die Memberfunktion definiert. Man übergibt Masse und Volumen und erhält den Quotienten zurück - die Dichte.

//10 An dieser Stelle deklarieren Sie eine Strukturvariable liquid1.

//11 An dieser Stelle deklarieren Sie eine zweite Strukturvariable liquid2. Mit dieser Variablen beschreiben Sie eine zweite Flüssigkeit.

//12 Nun definieren Sie die erste Strukturvariable. Zuerst definieren Sie die Masse der Flüssigkeit liquid1. Sie beträgt 1; z.B.: 1 Kg.

//13 Als nächstes definieren Sie das Volumen. Die Flüssigkeit wird Ihnen in einem Behälter gereicht, der einen Liter fasst. Das Volumen ist also auch 1.

//14 Und irgendwann möchten Sie die Dichte ermitteln. Dazu weisen Sie dem Funktionspointer als nächstes die Funktion retDichte zu.

//15 Anschließend weisen Sie dem Feld liquid1.cFarbe das Wort ”transparent” zu; das ist die Farbe der Flüssigkeit.

//16 Und in dem Feld liquid1.cFormel sichern Sie das Wort ”H2O”. Bei der Flüssigkeit, die mit der Strukturvariable liquid1 beschrieben wird handelt es sich um Wasser.

//17 An dieser Stelle geben wir Formel und Farbe der Flüssigkeit liquid1 aus.

//18 Hier nutzen wir die Funktion, die wir in der Struktur tLiquid deklariert haben und berechnen mit dem Members dMasse und dVolumen der Flüssigkeit liquid1 und der Memberfunktion dPtr2RetDichte die Dichte der Flüssigkeit. Wir können also mit einer einzigen Strukturvariablen Membervariablen UND Memberfunktionen zugleich benutzen. Das erleichtert uns durchaus den Überblick und sichert über Kompaktheit die Transparenz des Codes.
Das ist ein wesentlicher Erfolg und ein wichtiger Schritt in Richtung OOP.

//19 Anschließend geben wir den Namen der Flüssigkeit aus.

//20ff Nun wiederholen wir den ganzen Vorgang für die Flüssigkeit Benzol.

//21 Das Programm wird mit return 0 beendet.

Abbildung 2.5: Die Ausgabe des Programmes P001K2_C2CPP.

Wir wollen an dieser Stelle nochmals auf die Definition der Funktion retDichte eingehen. Gleich zu Beginn des Buches wollen wir uns zwei wichtige Dinge merken:

1. Jede Funktion sollte mit einem Kopfkommentar versehen sein
2. Sie sollten für mögliche ”Singularitäten” Vorsorge getragen werden.

Zu diesen zwei Punkten können Sie sich den Code 2.2 anschauen. Dazu die folgenden Bemerkungen:

zu 1.:

Neben den in C++ (und seit C99 auch in C) üblichen Einzeilenkommentaren, die mit // eingeleitet werden, sind auch Mehrzeilenkommentare möglich. Ein Mehrzeilenkommentar beginnt mit /* und endet mit */. Diese Kommentare können über mehrere Zeilen gehen. In unserem Beispiel geht der Kommentar nur über eine Zeile. Wir benutzen ihn, um einen Funktionskopf in geeigneter Weise zu kommentieren. Sie sollten diese oder eine ähnlich Kommentierung Ihrer Funktionen immer vornehmen. Es erleichtert Ihnen und anderen, den Code lesen zu können.

zu 2.:

Bitte beachten Sie, dass Sie mögliche Fehler innerhalb der Funktion abfangen. In dem Beispiel 2.2 vermeiden wir eine Division durch 0. Zusätzlich sollte eine Dichte immer größer Null sein. Dies wird mit

```
fabs
```

erreicht. `fabs` gibt den Absolutwert des Argumentes zurück. Dazu muss man allerdings math.h includieren:
`#include <math.h>`
Sollten Sie den Inklude von math.h vergessen, so sendet Ihnen der Compiler ein warning; siehe

```
/*=========================================================================*/
/* Desc.: Ermittelt aus Masse und Volumen die Dichte eines Koerpers.       */
/* In   : dMasse (double): Masse                                           */
/*        dVolumen (double): Volumen                                       */
/* Out  : Dichte (double);  Falls dVolumen <= 0 wird 0 zurueck gegeben.    */
/*=========================================================================*/
/*-------------------------------------------------------------------------*/
double retDichte (double dMasse, double dVolumen)                    //9
/*-------------------------------------------------------------------------*/
{
    if (dVolumen <= 0)
        return .0;
    return fabs(dMasse/dVolumen);
}
```

Programm 2.2: Die Funktion retDichte mit Kommentaren.

```
warning C4013: 'fabs' undefiniert; Annahme: extern mit Rückgabetyp int
```

Compilerausgabe 2.1: Compiler-warning bei vergessenem Inklude von math.h.

Compilerausgabe 2.1. Und wenn Sie diese Warnung nicht beherzigen, so werden Sie auf der Konsole sehr wirre Dichte-Werte sehen.

2.4 Zusammenfassung

- Wir haben gelernt, wie man mit dem Visual Studio Express 2012 ein Projekt anlegt.
- Jede Aufgabe sollte als ein separates Projekt behandelt werden. Bei größeren Aufgaben/Projekten sollte man zur Codeverwaltung ein Sourcecodeverwaltungs-Tool verwenden. Dazu empfehlen wir - weil es ganz einfach aus einem ”Guss” ist - Visual SourceSafe. Aus Zeit- und Platzgründen können wir hier allerdings nicht darauf eingehen.
- Während des Editierens Ihres Programmes können Sie sehr leicht C++ Schlüsselwörter (blau), Kommentare (grün) und Strings (magenta) voneinander unterscheiden.
- Eine gute Dokumentation des Programmes ist unabdinglich für ein späteres Verständnis der Software.
- Programmkommentare sollten kurz, treffend und in einer guten Form im Quellcode enthalten sein.
- Die (vorerst) wichtigsten Tasten im Visual Studio sind: F7 (Build), F9 Setzen/Löschen eines breakpoints, F5 Debugger starten, Shift F5 Debugger beenden.
- Zum Debuggen einer Aufgabe erstellen Sie eine Debugversion Ihres Programmes.
- Einzeilenkommentare werden // realisiert.
- Mehrzeilenkommentare werden mit /* ... */ realisiert. Hinweis: Vermeiden Sie geschachtelte Kommentare (/* .. /* .. */ .. */).
- Ein CPP-Programm wird im Visual Studio mit cl.exe generiert .
- Mit ”Gehe zu Disassembly” kann man den Assemblercode der Source während einer Debugsession kontrollieren.
- `struct` ist das zentrale Schlüsselwort, um den Übergang zu CPP verstehen zu können. Wir werden auf das Programm 2.1 bei der Einführung des Schlüsselwortes `class` zurückkommen. In Abbildung 2.6 ist der Versuch dargestellt, den Übergang von C nach C++ zu skizzieren.

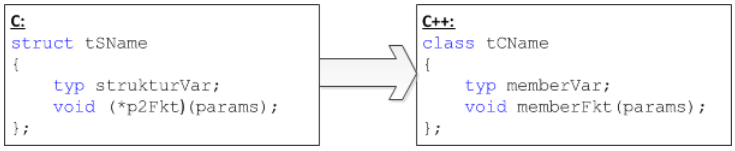


Abbildung 2.6: Der Übergang von C nach C++.

Zuletzt sei noch eine Bemerkung zu den Ausgaben des Compilers gestattet:
Ein Programm gilt dann als fehlerfrei übersetzt, wenn Sie 0 Fehler und 0 warnings haben. In der Compilerausgabe 2.2 kann man ein sauber übersetzbares Programm P001 feststellen.
Bitte beachten Sie, dass Ihre Software unbedingt warning-frei übersetzbar ist. In der Regel kann nämlich jedes warning auch einen möglichen Fehler verursachen! Und wenn Sie in einem großen Team an einem großen Projekt arbeiten, kann Ihr Kollege nicht wissen, ob Sie ein bestimmtes warning nur toleriert haben, oder ob es vergessen wurde, dieses warning zu beseitigen. Wenn man das Axiom, nur warning-frei übersetzbare Codes freizugeben, nicht konsequent einhält, so wird man sehr schnell während des Gesamtbuilds eines großen Software-Paketes dutzendweise warnings haben. Und jedes einzelne Teammitglied wird beteuern, dass seine warnings ungefähr-

lich sind und trotzdem kracht die Software an allen Ecken und Enden. Dann ist guter Rat im wahrsten Sinne des Wortes teuer!

```
===== Erstellen: 1 erfolgreich, Fehler bei 0, 0 aktuell, 0 übersprungen =====
```

Compilerausgabe 2.2: Compiler-Ausgabe bei P001 (Ausschnitt).

2.5 Übungen

1. Wir haben im Programm die Funktion retDichte wie im Code 2.2 zum Programm 2.1 hinzugefügt. Was haben wir falsch gemacht, wenn wir eine derartige Konsolenausgabe wie in 2.7 erhalten?
2. Welche Art von Kommentaren ist Ihnen bekannt?

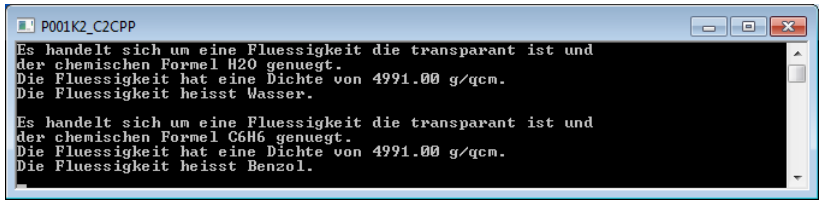


Abbildung 2.7: Die Konsolenausgabe des Programmes 2.1 mit geänderter Funktion retDichte wie im Code 2.2.

http://www.springer.com/978-3-658-18122-2

Softwareentwicklung mit C++
Einführung mit Visual Studio
Duschl, D.
2017, IX, 403 S. 226 Abb. in Farbe., Softcover
ISBN: 978-3-658-18122-2