Programmieren in

Programmieren lernen von Anfang an

>> Mit vielen Programmierbeispielen
>> Geeignet zum Selbststudium

Heimo Gaicher

Programmieren in

Programmieren lernen von Anfang an

>> Mit vielen Programmierbeispielen
>> Geeignet zum Selbststudium

Heimo Gaicher

Heimo Gaicher

Programmieren in C

Heimo Gaicher, Jahrgang 1969, ist bereits seit jungen Jahren leidenschaftlicher Bastler, Elektroniker und Programmierer. Berufsbegleitend absolvierte er nach seiner Lehre als Betriebselektriker die Werkmeisterschule für industrielle Elektronik und anschließend die Höhere Technische Bundeslehr- und Versuchsanstalt der Fachrichtung Elektronik und Technische Informatik in Graz. In seiner Industrielaufbahn befasst er sich mit der Hard- und Softwareentwicklung für innovative Produkte aus dem Bereich der HF-Kommunikations- und LED-Lichttechnik.

Vorwort

Dieses Buch richtet sich an Programmier-Einsteiger, die mit der Sprache C eine universelle Programmiersprache erlenen möchten. Da es für Einsteiger nicht gerade einfach ist, eine Programmiersprache zu erlernen, wurde dieses Buch schrittweise mit vielen Beispielen ausgearbeitet.

Auf eine allgemeine Einführung wurde bewusst verzichtet. Die entsprechenden Beispiele werden auch im Programmiercode genau dokumentiert und führen so zu einem besseren Verständnis. Das Buch ist kein Kompendium sondern eine für den Studierenden schrittweise Anleitung. Arbeiten Sie dieses Buch von Anfang an genau durch und versuchen Sie auch die Übungsaufgaben selbständig zu lösen. Denn eine Programmiersprache lernt man am besten, indem man Programme programmiert.

Dieses Buch wurde im Zuge meiner eigenen Ausbildung mit großer Sorgfalt geschrieben. Sollten sich dennoch Fehler eingeschlichen haben, freue ich mich über entsprechende Hinweise. Anmerkungen und Kritiken sind unter info@c-programmierung.at immer willkommen.

Viel Spaß beim Erlernen von C!

Autor: Heimo Gaicher
Umschlaggestaltung, Illustration: Patrick Gaicher

Verlag: tredition GmbH, Hamburg
ISBN: 978-3-8491-1848-8

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Inhalt

1. Einführung

Die Programmiersprache C wurde bereits 1972 von Dennis M. Ritchie in den Bell-Laboratorien (USA) entwickelt und 1973 bis 1974 von Brian W. Kernighan weiter verbessert. Als C ausgereift war und auch über eine Funktionsbibliothek verfügte, wurde die Sprache 1978 von Kernighan und Ritchie veröffentlicht.

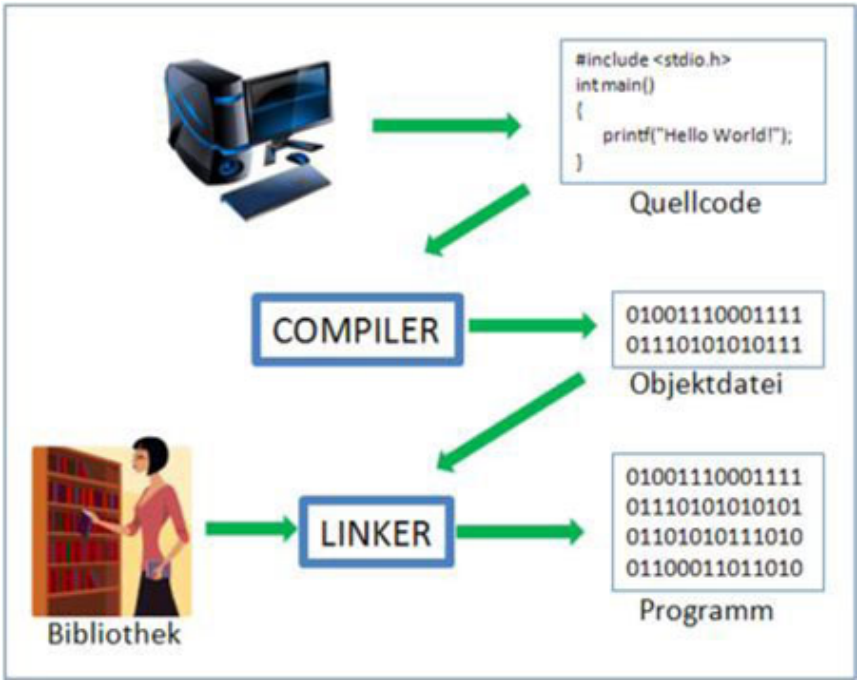
Die rasche Verbreitung von C verlangte schließlich eine Standardisierung. Diese wurde 1989 vom ANSI-Komitee festgelegt und 1990 von der ISO übernommen. Die Vorläufer von C sind die Sprachen BCPL (Basic Combined Programming Language) und B. Da die Sprache auf UNIX entwickelt und UNIX mit seinen Dienstprogrammen nahezu vollständig in C geschrieben wurde, scheint die Sprache sehr eng mit dem Betriebssystem UNIX verbunden zu sein. Doch C ist absolut universell einsetzbar und eine sehr „kleine“ aber doch mächtige Programmiersprache, die über wenige Anweisungen verfügt. Mit diesen wenigen Anweisungen deckt sie aber trotzdem alle geforderten Kontrollstrukturen ab.

Dadurch, dass viele Aufgaben wie z.B. Ein- und Ausgabe über vorgegebene Bibliotheksfunktionen gelöst werden, konnte die Anzahl der Anweisungen in C so gering gehalten werden. Aber auch sehr große Softwareprogramme wie z.B. das Betriebssystem Windows XP wurden größtenteils in C programmiert.

Doch auch in der hardwarenahen Programmierung ist C eine dominierende Programmiersprache. Auch für moderne Sprachen wie beispielsweise Java, C++ oder C# bildet C die Grundlage. Somit findet der C-Programmierer auch einen leichten Einstieg in diese Sprachen.

1.1 Wie erstellt man ein C-Programm?

Die Programmerstellung in C durchläuft prinzipiell drei Arbeitsschritte:
1. Erstellen des Quellcodes (editieren)
2. Übersetzen des Quellcodes (kompilieren)
3. Binden des kompilierten Codes (linken)



Der Quellcode wird entweder in einem Texteditor oder in einer Entwicklungs-umgebung, kurz IDE (**I**ntegrated **D**evelopment **E**nvironment) geschrieben. Dieser Programmcode ist für den Programmierer leserlich und verständlich. Doch für den Computer (Prozessor, Mikrocontroller) muss dieser Code in die sogenannte Maschinensprache (bestehend aus Nullen und Einsen) umgewandelt werden. Diese Umwandlung oder Übersetzung des Programmiercodes übernimmt ein Compiler, welcher in einer Entwicklungsumgebung bereits als Programm eingebunden ist.

Zum Compiler gehören auch mehrere Bibliotheken (Libraries). Die Bibliotheken enthalten bereits fertige Funktionen, welche Sie beim Programmieren in Ihre Programme einbinden können.

Der Linker sucht aus den Bibliotheken alle Funktionen die benötigt werden heraus und fügt sie dem Programm hinzu.

Betriebssysteme wie Linux oder Unix verfügen bereits automatisch über alle notwendigen Programme. Bei anderen Betriebssystemen wie z.B. Windows, muss eine Entwicklungsumgebung (IDE) für C-Programme gesondert installiert werden. Nur dann können Sie Ihren programmierten C-Code ausführen.

Um in C programmieren zu können, benötigen Siealso eine Entwicklungsumgebung. Natürlich können Sie ein C-Programm auch mit einem einfachen Texteditor schreiben, aber mit einer Entwicklungsumgebung ist die Programmierung wesentlich komfortabler. Wir verwenden in diesem Lehrbuch die Entwicklungsumgebung VISUAL C++ 2010 Express Edition, welche von Microsoft als Freeware angeboten wird. Alternativ kann aber auch jede andere Entwicklungsumgebung (z.B. **Dev-C++** oder **wxDev-C++**) für C verwendet werden.

1.2 Installation von VISUAL STUDIO Express 2010

VISUAL STUDIO 2010 EXPRESS EDITION ist eine kostenlose Software und Sie können Sie direkt von der Microsoft-Webseite herunterladen.

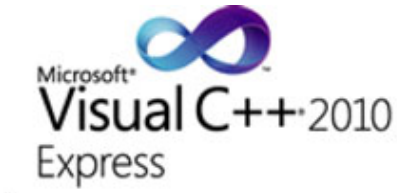
Hier der Downloadlink:

http://www.microsoft.com/germany/express/download/webdownload.aspx

1.3 Download und Installation unter Windows

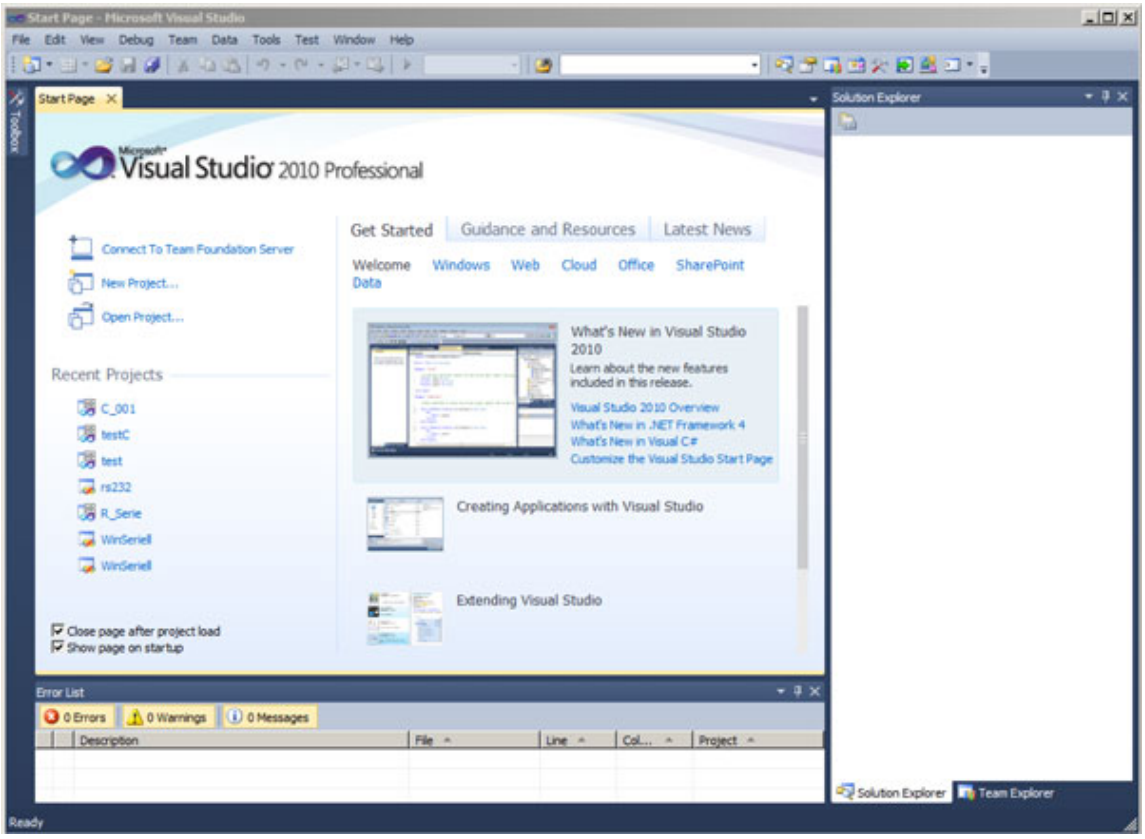
VISUAL STUDIO EXPRESS einrichten

Wählen Sie **VISUAL C++ 2010EXPRESS** und klicken Sie auf Download. Wählen Sie bei der Installation alle Optionen aus. Zum Zeitpunkt der Erstellung dieses Buches ist die Version 2010 aktuell.

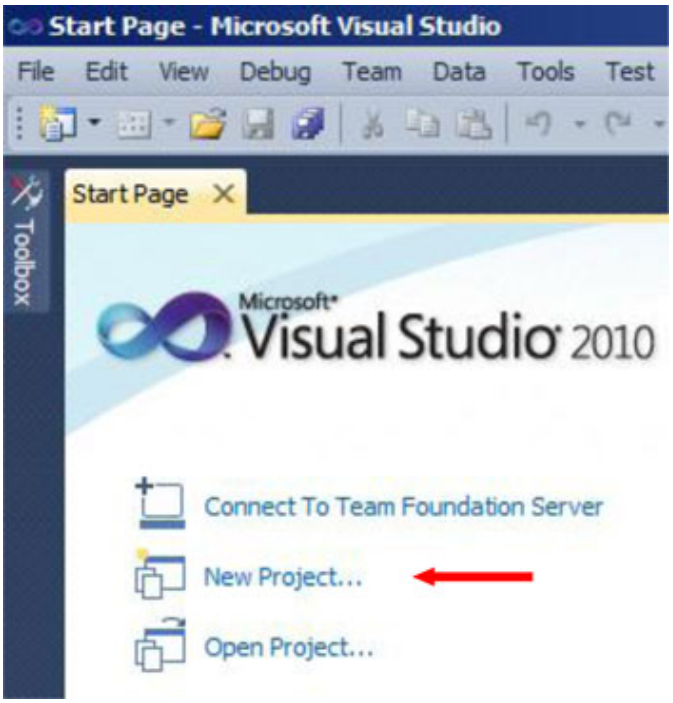


1.4 Start mit VISUAL STUDIO Express 2010

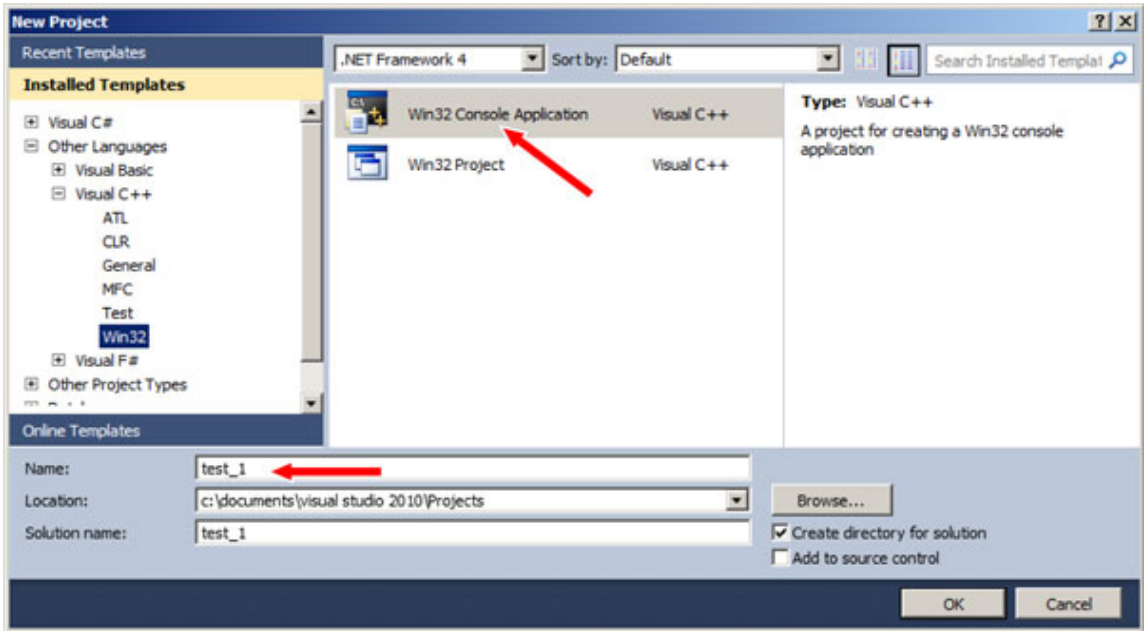
Starten Sie VISUAL C++ 2010 EXPRESS.
Sie gelangen zum Startfenster, welches beim Start in etwa wie in der folgenden Abbildung aussieht.



Klicken Sie nun auf der Startseite auf „Neues Projekt“.

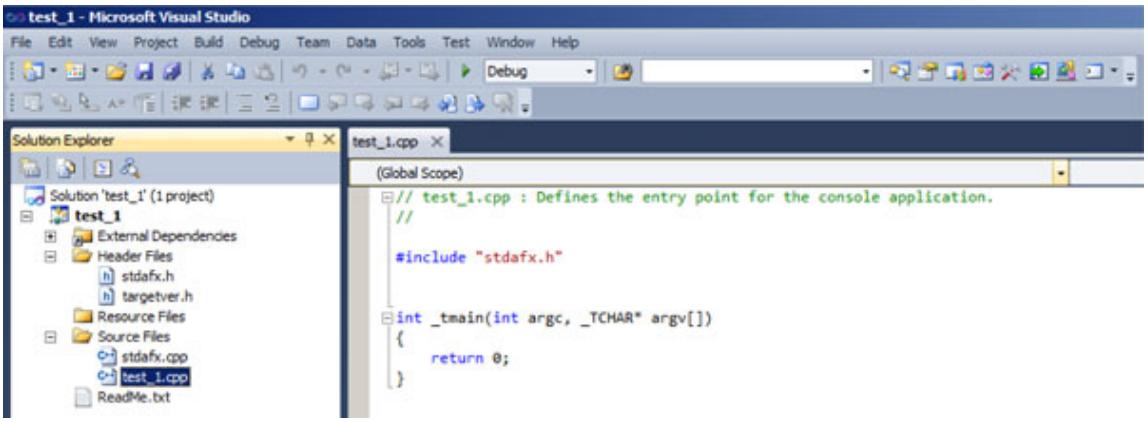


Um eine Konsolenanwendung zu starten, verwenden Sie die Win32-Konsolenanwendung und geben einen Namen für Ihr Projekt ein.



Klicken Sie auf „OK“ und nachfolgend auf „Fertigstellen“.

Jetzt haben Sie ein neues Projekt erstellt und können im Editor Ihren Programmcode in C++ oder wie in unserem Fall in C programmieren. Die Quelldatei für unseren Programmcode heißt **test_1cpp**.



In VISUAL C++ lässt sich die Anordnung der Fenster flexibel gestalten. Sämtliche Fenster lassen sich an allen Ecken andocken. So können Sie sich VISUAL C++ nach Ihren eigenen Bedürfnissen einrichten. Sie werden im Laufe der Programmierübungen noch einiges über den Umgang mit VISUAL Studio lernen.

1.5 Das erste C-Programm

Testen wir nun unsere Entwicklungsumgebung und schreiben ein erstes Programm. Fügen Sie dazu unter der Startfunktion *_tmain()* die Programmzeile `printf("Ich bin ein C Fan!");` ein. Mit *printf()* wird der Text zwischen den beiden Anführungszeichen am Bildschirm ausgegeben.

Als Anfänger werden Sie an dieser Stelle den kompletten Programmcode noch nicht verstehen. Das ist aber zu diesem Zeitpunkt auch nicht notwendig, denn wir werden uns Schritt für Schritt an die Funktionen in C herantasten.

```
int _tmain(int argc, _TCHAR* argv[])
{
  printf("Ich bin ein C Fan!");
```

```
  return 0;
}
```

Im nächsten Schritt möchten wir unser Programm ausführen. Dazu starten Sie den Debugger indem Sie die Taste **F5** drücken oder das Symbol (**>**)anklicken.

Jetzt wird der Programmcode zuerst kompiliert und wenn der Compiler keinen Fehler erkennt, wird der Programmcode im Debug-Fenster ausgeführt. Wenn Sie den Programmcode ausführen, werden Sie feststellen, dass sich das Debug-Fenster ganz kurz öffnet und gleich wieder schließt. Damit das Debug-Fenster geöffnet bleibt, müssen wir unser Programm vor dem Programmende anhalten. Dies können wir z.B. mit der Funktion *getchar()* realisieren. Diese Funktion wartet an dieser Stelle auf eine Tastatureingabe, welche mit der Taste „Enter“ abgeschlossen wird.

Bauen wir also die Funktion *getchar()* in unser Programm ein.

```
int _tmain(int argc, _TCHAR* argv[])
{
printf("Ich bin ein C Fan!");
getchar(); //An dieser Stelle auf eine Tastatureingabe warten
return 0;
}
```

Starten Sie den Debugger erneut. Wenn Sie alles richtig gemacht haben, bleibt das Debug-Fenster nun geöffnet und die Ausgabe <**Ich bin ein C Fan!>** erscheint am Bildschirm.

Wie Sie im Codebeispiel sehen, wurde mit **//** ein Kommentar eingeleitet. Kommentare sind für den Programmierer eine wichtige Möglichkeit um Programmcode zu dokumentieren.

Insbesondere bei größeren Programmen wird eine gute Dokumentation sehr wichtig, da Sie dem Programmierer hilft, auch nach längerer Zeit, den Programmcode sehr rasch zu

verstehen. Kommentare werden vom Compiler nicht berücksichtigt.

Betrachten wir unseren Programmcode nun etwas genauer.Unser Projekt (auch jedes andere) startet an der Stelle **_tmain** und wird von den beiden geschwungenen Klammern { } eingeschlossen.

```
int _tmain(int argc, _TCHAR* argv[])
{
//Hier steht der Code…
}
```

Innerhalb der beiden Klammern wird unser Code schrittweise abgearbeitet und das Programm entsprechend ausgeführt.Da wir unser erstes Projekt mit den Anwendungseinstellungen

- Konsolenanwendung
- Vorkompilierter Header

erstellt haben, wurden von der Entwicklungsumgebung automatisch sogenannte Headerdateien (das sind Dateien mit der Endung .h) vorgeladen. Diese Dateien z.B. **stdafx.h** ermöglichen eine sehr rasche Kompilierung des Quelltextes. Dadurch sinkt die Kompilierzeit für große Programme dramatisch.

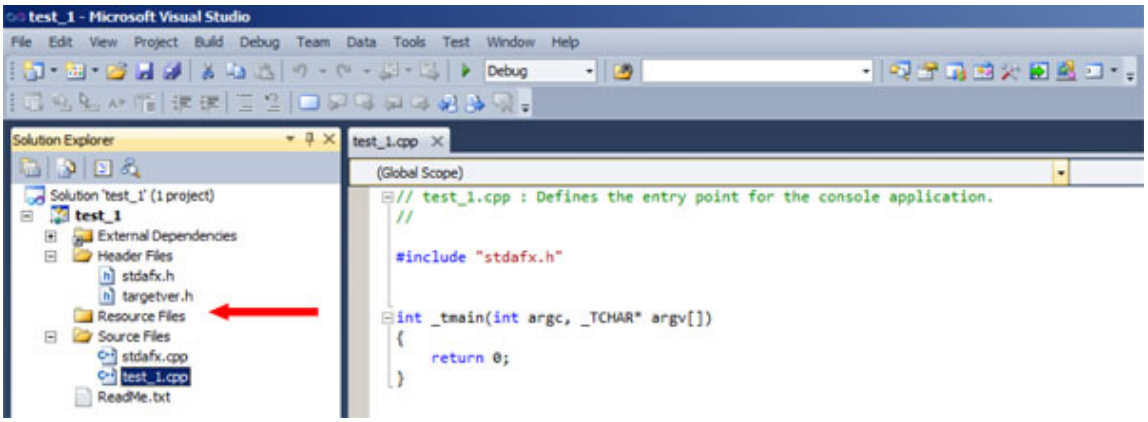
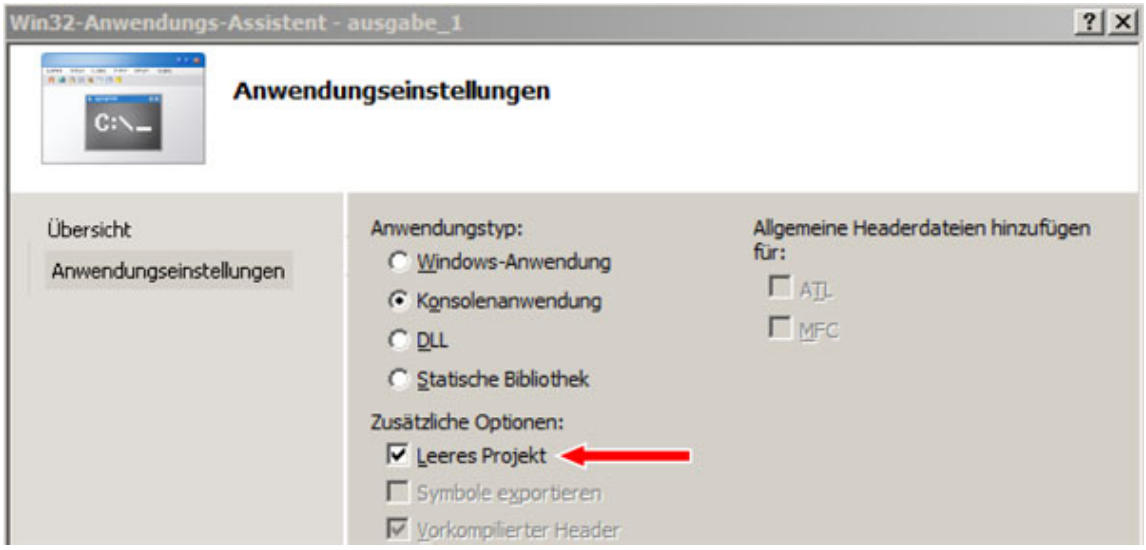


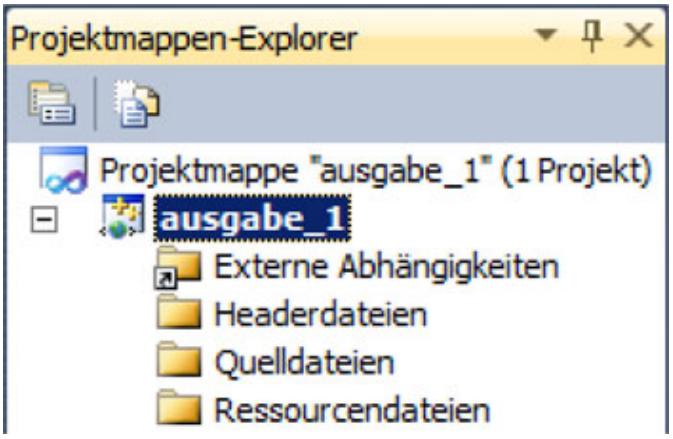
Abb: Durch vorkompilierte Header erzeugte Projektdateien

Da wir in diesem Lehrbuch aber keine großen Programme schreiben, werden wir der Überschaubarkeit wegen, unser Projekt etwas umgestalten.

Erstellen Sie ein neues Projekt mit dem Namen ausgabe_1 und ändern die Anwendungseinstellungen auf „Leeres Projekt“.

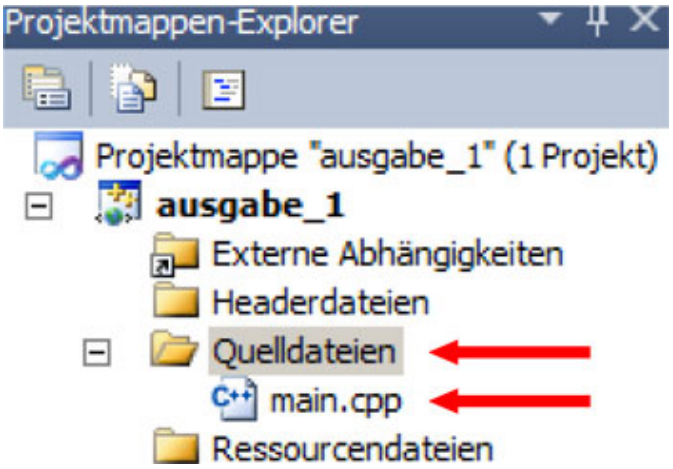


Es öffnet sich der Projektmappen-Explorer wie in der folgenden Abbildung ersichtlich mit einem leeren Projekt.



Um eine neue Quelldatei zu erstellen, klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf Quelldateien und wählen Hinzufügen /Neues Element.

Wählen Sie nun die installierte Vorlage Code / C++ Datei und geben Sie der Datei den Namen **main**. Der Projektmappen-Explorer hat nun die Datei **main.cpp** im Verzeichnis Quelldateien erstellt. In diese Datei schreiben wir unseren Programmcode.



Geben Sie nun den folgenden Code im Editor ein und starten den Debugger:

```
int main(void)
{
printf("Ich bin ein C-Fan!");
getchar();
return 0;
}
```

Im Ausgabefenster erhalten Sie jetzt zwei Fehlermeldungen:

```
"printf": Bezeichner wurde nicht gefunden.
"getchar": Bezeichner wurde nicht gefunden.
```

Der Compiler kann in diesem Moment die beiden Funktionen *printf()* und *getchar()* nicht zuordnen. Diese Standardfunktionen müssen dem Compiler am Anfang „bekannt“ gemacht werden. Dies geschieht durch das Einbinden der Headerdatei ***<stdio.h>***.

Eine Headerdatei wird mit *#include* eingebunden.

```c
#include <stdio.h> // Bindet die Standard Ein- und Ausgabefunktionen ein
int main(void)
{
printf("Ich bin ein C-Fan!");
getchar();
return 0;
}
```

Bei erfolgreicher Kompilierung wird das Konsolefenster geöffnet und das Programm ausgeführt.

In der Konsole wird

ausgegeben.

Zum Unterschied von vorher verwenden wir als Einstiegspunkt nur die leere Funktion *main()* ohne die Übergabeparameter `(int argc, _TCHAR* argv[])`. Welche Bedeutung Übergabeparameter haben werden Sie zu einem späteren Zeitpunkt noch kennenlernen.

Dasselbe Programm könnten Sie nun auch mit einem einfachen Texteditor schreiben und als „Dateiname.c“ speichern.

Um ein Programm an bestimmter Stelle anzuhalten können Sie auch die Funktion *system(“PAUSE“)* verwenden. Diese Funktion befindet sich in der C-Standardbibliothek *stdlib.h* welche zuvor mit *#include* eingebunden werden muss.

```c
#include <stdlib.h> // Bindet die C-Standardbibliothek ein
system("PAUSE"); // Hält ein Programm an dieser Stelle an bis eine Taste gedrückt wird
```

1.6 Startfunktionen in C

Mit `#include <stdio.h>` (Präprozessor-Direktive) wird eine Bibliothek, in welcher Standardfunktionen hinterlegt sind, in das Programm

geladen. Mit `stdio.h` werden Makros und Variablen definiert, die in der Standardbibliothek verwendet werden. Sie wird für die Standard-Ein und Ausgabe (standard input / output) benötigt. Wenn der Compiler die Anweisung verarbeitet hat, ist diese Datei ein fixer Bestandteil des Programms.

`int main(void)`ist der **Startpunkt** eines jeden C-Programms.

Die Funktion *main()* muss in jedem Programm **einmal** vorkommen. Sie darf aber auch nicht öfter als einmal vorkommen. Zu jeder Funktion gehört auch ein Funktionsrumpf, welcher durch die beiden geschwungenen Klammern **{ }** gekennzeichnet ist. Alles, was innerhalb der beiden geschwungenen Klammern steht, gehört auch zu dieser Funktion.

In unserem Beispiel haben wir den Funktionsaufruf `printf("Ich bin ein C-Fan!");` verwendet. Mit der Funktion *printf()* wird am Bildschirm der gesamte Inhalt zwischen den beiden Klammern ( ) ausgegeben. Also alles, was innerhalb der beiden Anführungszeichen steht. Das **Semikolon (;)** am Ende des Befehls ist zwingend erforderlich und schließt den Befehl ab.

Sehen wir uns den Startpunkt *main()* noch etwas genauer an:
`int main(void)`

Das Schlüsselwort *void* verwendet man, um Funktionen zu deklarieren, die **keine Ergebnisse** enthalten sollen. Void bedeutet also ganz einfach leer.

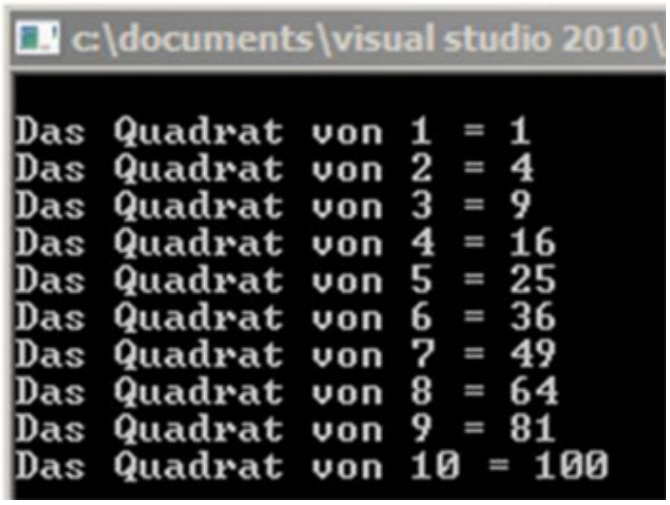
Mit `int main(void)` wird kein Parameter übergeben aber ein Parameter (Rückgabeparameter) erwartet. Was das konkret bedeutet, werden wir an späterer Stelle noch klären. Derzeit ist nur wichtig, dass Sie wissen, dass der Einstiegspunkt für jedes Programm die Funktion *main()* ist. Wird kein Parameter zurückgegeben, ist auch die Deklaration `void main(void)` möglich. In diesem Fall entfällt `return 0;` am Ende des Programms.

1.7 Einfache Programmbeispiele

Die folgenden Programmbeispiele dienen lediglich der Einführung in die Programmierung, damit Sie mit den Codebeispielen etwas vertraut werden um die Theorie leichter zu verstehen. Es ist nicht erforderlich, dass Sie den Code bereits jetzt verstehen. Vieles davon ist jedoch selbsterklärend und alles andere wird an späterer Stelle behandelt werden.

1.7.1 Berechnung und Ausgabe von Quadratzahlen

Es soll ein Programm geschrieben werden, welches die ersten zehn Quadrate (also die Quadrate von 1 bis 10) berechnet und ausgibt.




```c
#include <stdio.h>
int main(void)
{
int zahl=1, ergebnis;
for (int i=0; i<=9; i++)
{
ergebnis = zahl * zahl;
printf("\nDas Quadrat von %i = %i",zahl, ergebnis);
zahl++;
}
getchar();
```

```
return 0;
}
```

Erläuterungen zum Programmcode:

Wie Sie bereits wissen, muss immer ein Hauptprogramm vorhanden sein. Das Hauptprogramm ist sozusagen der Startpunkt für die Programmausführung. Das Hauptprogramm in C heißt immer *main()*. Zuvor ist die Präprozessor-Anweisung *#include<stdio.h>* erforderlich damit die Bibliotheksfunktion *printf()* erkannt wird.

In der nächsten Programmzeile werden zwei Variablen (zahl und ergebnis) deklariert. Beide Variablen sind vom Datentyp Integer (Integer = ganze Zahlen) wobei der Variablen zahl bereits ein Wert (zahl=1) zugewiesen wird.

Jetzt folgt eine for-Schleife. Eine for-Schleife wiederholt den Programmcode innerhalb der beiden geschwungenen Klammern { } so oft, wie dies in der Schleife definiert wurde. In diesem Fall wird eine Zählvariable i vom Datentyp Integer vereinbart. Der Startwert von i ist 0. Der Endwert von i ist 9. Mit i++ wird die Zählvariable i bei jedem Schleifendurchlauf um eins erhöht. Die Schleife wird also so lange durchgeführt, so lange die Zählvariable i kleiner oder gleich 9 ist. Ist i gleich 9, wird die Schleife noch ein einziges Mal durchlaufen und die Zählvariable auf 10 erhöht. Danach wartet das Programm mit *getchar()* auf eine Tastatureingabe und wird beendet.

Im ersten Schleifendurchlauf erhält die Variable ergebnis den Wert 1*1 und wird mit *printf()* am Bildschirm ausgegeben. Anschließend wird die Variable zahl um eins erhöht. Im zweiten Schleifendurchlauf erhält die Variable ergebnis den Wert 2*2 und wird mit *printf()* am Bildschirm ausgegeben. Anschließend wird die Variable zahl wieder um eins erhöht. Dieser Vorgang setzt sich solange fort, solange die Schleifenbedingung (i <= 9) erfüllt ist.

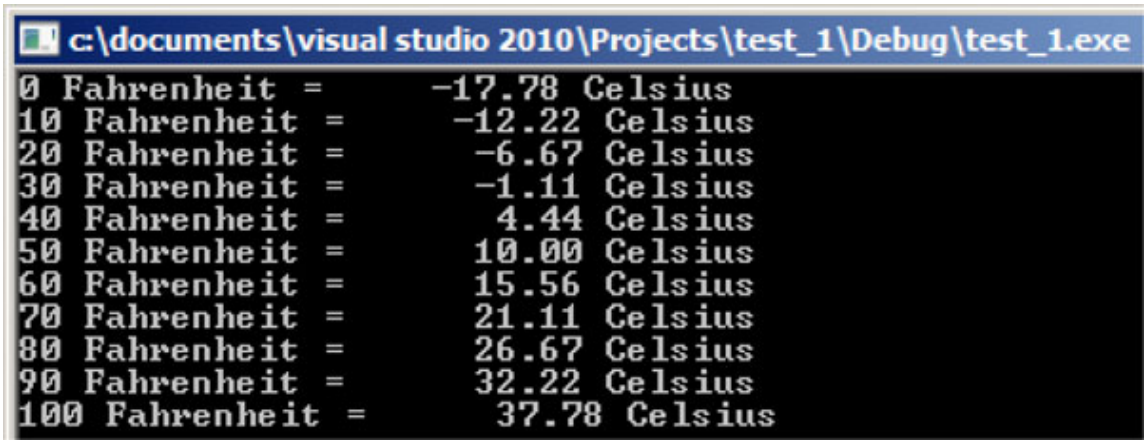
Nicht zu vergessen ist das Semikolon (**;**). Das Semikolon ist in C ein Satzzeichen und wird dazu verwendet, das Ende einer Anweisung anzuzeigen. Eine Anweisung kann sich in C auch über mehrere Zeilen erstrecken.

1.7.2 Umrechnung von Fahrenheit in °C

Mit diesem Programm werden Temperaturen in Zehner-Schritten von 0 bis 100 Grad Fahrenheit in Grad Celsius umgerechnet und als Tabelle ausgegeben.

Die Formel für die Umrechnung lautet: $°C = \frac{5}{9} * (°F - 32)$

```
#include <stdio.h>
int main(void)
{
int fahrenheit = 0;
double celsius;
while (fahrenheit <= 100) // Führe durch, solange fahrenheit kleiner
oder gleich 100
{
celsius = ((5.0/9.0)*(fahrenheit-32)); // Division von 5.0 / 9.0
// Ergebnis als Gleitkomma
printf("%i Fahrenheit = %10.2lf Celsius\n",fahrenheit, celsius);
fahrenheit +=10; // Erhöhe fahrenheit um 10
}
getchar();
return 0;
}
```



Erläuterungen zum Programmcode:

Das entscheidende in diesem Programm ist die Division. Würde man 5/9 schreiben, wäre das Ergebnis 0 da eine Integer Division (ganzzahliger Datentyp) vorliegt. Hier ist aber eine Division mit Rest (Gleitpunkt Division) gefordert. Mit 5.0 / 9.0 erzwingt man die Durchführung einer Gleitpunkt Division.

Das Ergebnis soll um zehn Stellen nach rechts verschoben mit einer Genauigkeit von zwei Dezimalstellen angezeigt werden. Das **Formatelement** lautet daher **%10.2lf**. 10 steht für die Verschiebung nach rechts um zehn Stellen. 2 steht für die Anzahl der Nachkommastellen und lf für die Ausgabe einer Zahl vom Datentyp **double**.

1.7.3 Berechnung des Zinseszinseffektes

Das folgende Programm berechnet die Zinsen und Zinseszinsen auf eine einmalige Kapitalanlage und gibt die jährlichen Ergebnisse in Form einer Tabelle aus.

```
#include <stdio.h>
#define LAUFZEIT 10
#define STARTKAPITAL 10000
#define ZINSSATZ 3.25
int main(void)
{
double kapital = STARTKAPITAL;
for (int jahr=1; jahr <= LAUFZEIT; jahr++)
{
kapital = kapital * (1.0 + ZINSSATZ / 100.0);
printf("\nIhr Kapital nach dem %i jahr: %10.2lf ",jahr,kapital);
}
getchar();
return 0;
}
```