

Zeichen für Zeichen

Textdateien zeilenweise lesen und ausgeben

Bisher haben Sie das grundlegende Hintergrundwissen erworben, das Sie für die Programmierung von C(++)-Programmen benötigen: Nun geht's ans Eingemachte: Das erste echte Projekt soll her.

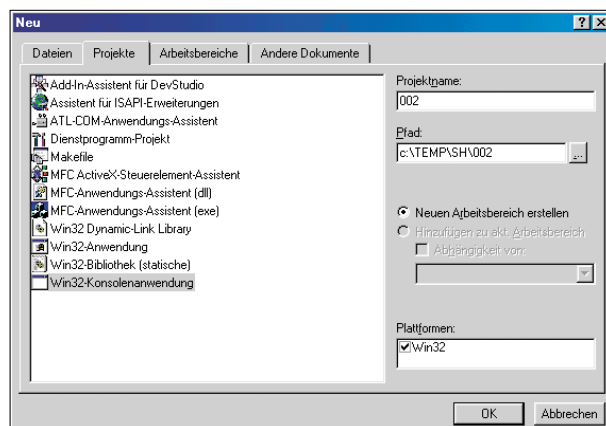
THOMAS WÖLFER

Zwar ist noch längst nicht der gesamte Sprachumfang dargelegt – aber alle Elemente, die Sie für die ersten Programme benötigen, sind vorgestellt. Beim ersten eigenen Beispielprojekt ist eine einfache Aufgabe zu erfüllen: Es soll ein Programm entwickelt werden, das in der Lage ist, Textdateien am Bildschirm anzuzeigen – und zwar die Datei, die dem Programm als Kommandozeilenparameter übergeben wird. Für diese relativ einfache Aufgabenstellung werden eine ganze Reihe Funktionen benötigt. Sie müssen die Kommandozeile auswerten, eine Datei zum Lesen öffnen, die Datei zeilenweise auslesen und das Gelesene schließlich am Bildschirm anzeigen. Praktischerweise liefert aber

die C(++)-Laufzeitbibliothek fast alle Funktionen, die dafür notwendig sind.

■ Der erste Schritt: Projekt anlegen

Zunächst brauchen Sie ein neues VC++-Projekt. Das legen Sie wie schon gewohnt mit dem passenden Assistenten an, und genau wie beim ersten Projekt wählen Sie eine Win32-Kommandozeilen-Anwendung als Zieltyp für Ihr



AUCH DIESES PROJEKT wird als Win32-Kommandozeilenprogramm realisiert.

Projekt aus. Auf der CD zu diesem Sonderheft finden Sie das komplette Projekt, einschließlich einer Testdatei zum Einlesen.

Um von diesem Beitrag am meisten zu profitieren, sollten Sie aber zunächst versuchen, das Programm selbst zu schreiben. Dabei helfen Ihnen die fol-

LISTING 1

```
int main ( int argc, char* argv[] )
{
    return 0;
}
```

genden Anleitungen – gegen Ende des Beitrages finden Sie dann eine Erläuterung eines komplett funktionstüchtigen Programms.

Der VC++-Assistent hat Ihnen einen Quellcode generiert, der im Wesentlichen die Zeilen enthält, die Sie in Listing 1 finden.

Da stellt sich die Frage, was das bedeutet. Zunächst haben Sie es hier mit der bereits bekannten *main()*-Funktion zu tun. Diese Funktion ist in jedem normalen C/C++-Programm vorhanden – an dieser Stelle beginnt der Lauf Ihres Programms. Wenn das Programm geladen wird, wird die Kontrolle an einen bestimmten Eintrittspunkt im Programm übergeben. Den richtigen Punkt – oder besser, die richtige Funktion dazu – kennt der Linker. Nun könnte man annehmen, dass dieser Eintrittspunkt ins Programm die Funktion *main()* ist – das ist aber nicht der Fall. Der Grund dafür ist der, dass die C++-Laufzeitbibliothek noch "vor" Ihrem eigenen Code drankommt. Denn diese Bibliothek enthält einige Variable und Datenstrukturen, die vor der ersten Benutzung initialisiert werden müssen. Nachdem Sie aber in der Funktion *main()* sofort damit beginnen können, die Funktionen der C++-Laufzeitbibliothek zu verwenden, ist es logischerweise notwendig, dass diese Initialisierung vor dem Aufruf von *main()* stattfindet.

Das ist auch genau das, was passiert. Beim Laden des Programms wird die Kontrolle zunächst an eine dem Linker bekannte Funktion aus der C-Laufzeit übertragen. Dort kann dann sichergestellt werden, dass die Laufzeitbibliothek korrekt initialisiert wurde. Darum müssen Sie sich als Entwickler nicht kümmern, denn dafür haben bereits die Programmierer bei Microsoft Sorge ge-

tragen, die für die Programmierung dieser Bibliothek zuständig sind.

Erst wenn diese Initialisierung abgeschlossen ist, ruft die Laufzeitbibliothek Ihre *main()*-Funktion auf: Der eigene Code wird gestartet. Dabei hat *main()* zwei Parameter: Einen vom Typ Integer mit dem Namen "argc" und einen vom Typ *char** mit dem Namen "argv". Mit dieser Variablen haben Sie es nun mit Ihrem ersten Zeiger zu tun. Doch eins nach dem anderen – argc kommt zuerst, und soll auch als erstes erläutert werden.

■ argc – ARGument Count

Der Name "argc" ist eine Abkürzung und steht für "ARGument Count" – es handelt sich dabei um eine Angabe, die die Anzahl der Argumente betrifft. Die Funktion *main()* wird im Normalfall die einzige Funktion sein, bei der Sie auf eine solche Angabe stoßen, denn nur hier ist die Anzahl an Parametern von Haus aus nicht bekannt.

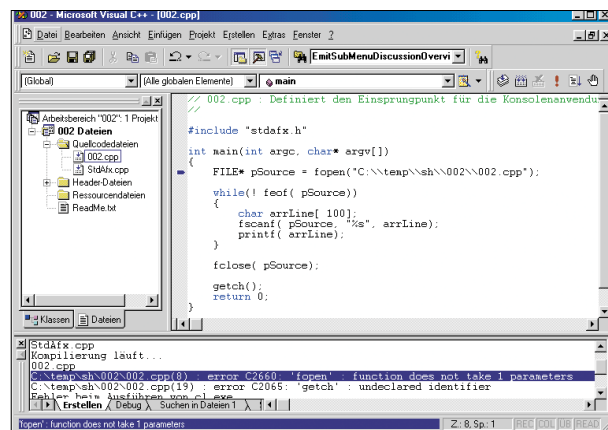
Das hat einen guten Grund: Wenn Sie ein Kommandozeilenprogramm starten, können Sie das unter Angabe von Parametern tun. Sie kennen solche Parameter vermutlich vom *del*-Befehl. Hier geben Sie als Parameter den Namen der zu lö-

Genau das Gleiche gilt natürlich auch für andere Programme, und so auch für das Ihre: angenommen, Ihr fertiges Programm hat den Namen 002.exe. In diesem Fall können Sie das Programm auf die unterschiedlichsten Arten aufrufen, zum Beispiel so:

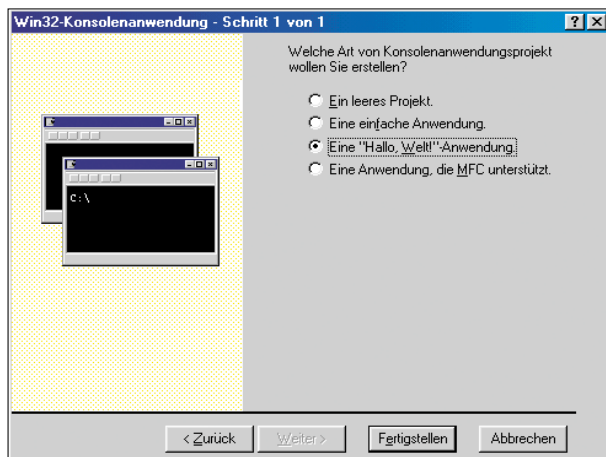
```
c:\> 002.exe NameDer
      ↳ZulöschendenDatei
```

oder aber auch so:

```
c:\> 002.exe Haribo
      ↳macht Kinder froh
```



WENN SIE SICH BEIM EINGEBEN des Quellcodes vertippen, zeigt der Compiler dies im "Erstellen"-Fenster an, sobald Sie den Übersetzungsvorgang starten.



ALS VORLAGE verwenden Sie eine "Hallo Welt"-Anwendung – den "Hallo Welt"-Teil können Sie entfernen, denn Sie brauchen nur die *main()*-Funktion ohne Inhalt.

schenden Datei an. Sie können aber auch mehrere Dateinamen angeben – und jeder dieser Dateinamen ist ja ein eigenständiger Parameter für den Befehl. Mit anderen Worten: Auch der *del*-Befehl weiß nicht, wie groß die Anzahl der übergebenen Parameter sein wird, denn das ist erst dann bekannt, wenn der Benutzer den kompletten Befehl eingetippt hat.

höher, als die Anzahl der Parameter auf der Kommandozeile.

■ Argv – der Argument- Vector

Der zweite Parameter hat eine bisher nicht angesprochene Schreibweise und einen ebenso ungewöhnlichen Namen wie der erste. "argv" steht für "ARGument Vector", und das bedeutet, die Va-

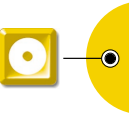
riable enthält einen Vektor für alle Argumente. Hier sind also alle ans Programm übergebenen Parameter enthalten, ganz gleich, wie viele das sind. Außerdem fallen die eckigen Klammern hinter dem Namen der Variable auf. Eckige Klammern werden in C/C++ für Arrays verwendet. Sie geben damit nicht nur an, dass es sich bei einer Variablen um eine Array handelt, sondern indizieren auch in solche. Sie geben also an, welche Komponenten Sie darin erreichen wollen.

Bei einem Array handelt es sich nämlich um eine Datenstruktur, bei der mehrere Elemente vom gleichen Datentyp hintereinander im Speicher stehen. So können Sie zum Beispiel ein Array aus Integer anlegen, das eine Gruppe von Integer aufnehmen kann. Das geht aber auch mit allen anderen Datentypen. Welchen Datentyp eine Variable hat, steht in C/C++ immer vor

dem Namen der Variablen: In unserem Fall – wenn Zeichen zu lesen sind – ist das also "char*". Wäre der lästige Stern hier nicht im Wege, dann wäre der Fall klar: Ein "char arg[]" wäre ein "Array" aus "char" – also eine Ansammlung von Zeichen (characters). Ein Character ist dabei ein ASCII-Zeichen und ein einzelnes dieser Zeichen kann die numerischen Werte zwischen 0 und 255 annehmen. (So hat der Buchstabe "A" zum Beispiel den Integer-Wert 65, der Buchstabe "B" den Wert 66 und so weiter.)

Der * gibt nun an, dass es sich eben nicht um eine "char"-Variable handelt, sondern um einen Zeiger auf char. Man sagt dazu "Es handelt sich um einen Pointer to Char (Zeiger auf Char)". Zeiger-Variablen können Sie von jedem Typ haben: So gibt es Pointer to Char, Pointer to Int, Pointer to Double und so weiter.

Eine Zeiger-Variablen ist dabei immer groß genug, um die Adresse einer Variablen des angegebenen Datentyps aufzunehmen. Wird Ihr Programm in den Arbeitsspeicher geladen, hat jede Variable eine klar definierte Speicherzelle, bzw. einen Speicherbereich. Diese Speicherzelle hat eine eindeutige Adresse.



Um das zu verstehen, stellt man sich den Arbeitsspeicher einfach als eine lange Kette von einzelnen Bytes vor. In dieser Kette hat jedes Byte – also jede Speicherzelle – eine eindeutige Nummer. Genau das ist die Adresse der Speicherzelle.

Angenommen Sie haben eine "char"-Variable, die den Buchstaben "A" (also den Wert 65) enthält, und diese befindet sich in der hundertsten Speicherzelle, dann hat diese Variable die Adresse "100". Mit einer Zeiger-Variablen können Sie nun diese Adresse speichern.

Sie haben dann eine Variable, deren Inhalt der Adresse der ersten Variable entspricht – man sagt dann "Sie zeigt auf die andere Variable". Dass es in C/C++ möglich ist, die Adresse einer Variablen in einer anderen Variablen abzuspeichern, ist ungeheuer praktisch – wie Sie in Kürze selbst feststellen werden.

Doch weiter mit "argv". Die Variable ist also eine Array-Variable vom Typ "Pointer to Char": Sie haben also ein Array von Zeigern auf Character vorliegen und die Größe des Arrays – also die Anzahl der Elemente darin – wird Ihnen durch den Wert der Variable "argc" mitgeteilt. Das bedeutet, dass argv[0],

sind in C/C++ immer Arrays aus Characters, die mit einem "0" Zeichen abgeschlossen werden, und die Adresse des ersten Elements dieses Arrays kann in einer Zeiger-Variablen abgelegt werden. Angenommen der Text "Hallo" wird im Arbeitsspeicher abgelegt, und angenommen der Buchstabe "H" befindet sich an der Adresse 100, haben Sie dann im Arbeitsspeicher das Layout, was Sie im Kasten "Layout des Arbeitsspeichers von "Hallo"" sehen.

An Adresse 100 befindet sich also der Buchstabe "H" mit dem Wert 100, an Adresse 101 – also ein Byte weiter – befindet sich der Buchstabe "A" mit dem Wert 65 und an Adresse 105 befindet sich das (symbolische) Zeichen "NUL" mit dem Wert "0". Alle Funktionen in C/C++, die mit Strings arbeiten, erwarten am Ende eines Textes dieses NUL-Zeichen – es markiert das Ende eines Textes.

Jedes an Ihr Programm übergebene Argument liegt nun in Form eines solchen Strings vor, und die Anfangsadressen dieser Strings sind im Array argv abgelegt. Das bedeutet, dass

ser Seite.) Wenn Sie also Ihrem Programm einen Dateinamen (die anzuzeigende Datei) als Parameter übergeben, taucht die Adresse des übergebenen Strings an der entsprechenden Stelle in argv auf.

Das ist das erste, was Sie nun einmal ausprobieren sollten: Zeigen Sie alle ans Programm übergebenen Parameter an. Dazu brauchen Sie eine kleine Schleife und den printf()-Befehl, den Sie bereits kennengelernt haben. (Vergessen Sie dabei nicht

LISTING 2

```
#include <conio.h>
int main( int argc, char* argv[])
{
    for( int i=0; i<argc; i++)
    {
        printf( argv[ i]);

    }

    getch();
    return 0;
}
```

"conio.h" und den getch()-Befehl, denn sonst geht das Programmfenster gleich wieder zu.) (Siehe dazu Listing 2.)

IDE-Probleme: Argumente übergeben

Wenn Sie dieses Programm nun übersetzen und ausführen (/F5), dann stellen Sie es schnell fest: Insgesamt gibt es nur einen Parameter – und das ist der bereits erwähnte vollständige Pfad auf Ihr Programm. Der gewünschte Pfad auf die anzuzeigende Datei fehlt aber – das ist auch kein Wunder, denn Sie haben ja gar keinen Parameter angegeben.

Trotzdem können Sie die Sache kurz testen, und zwar außerhalb der IDE. Öffnen Sie dazu ein Kommandozeilenfenster, und wechseln Sie in das Verzeichnis, in dem sich die soeben erstellte .EXE-Datei befindet. Wenn Sie diese aufrufen und dabei gleichzeitig auch als Parameter übergeben, werden zwei Parameter (zweimal der Pfad) am Bildschirm angezeigt. Zwar sind sie noch ein bisschen un-

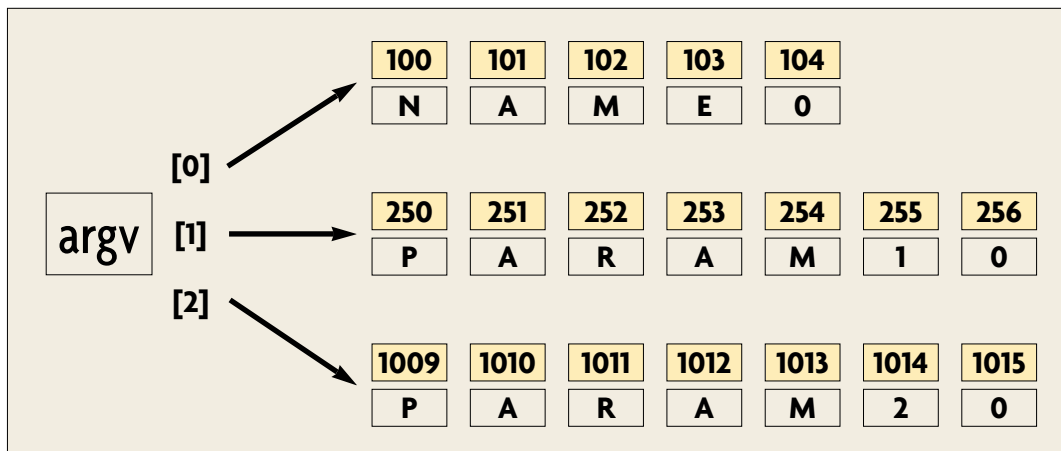
LAYOUT DES ARBEITSSPEICHERS VON "HALLO"

Adresse:	100	101	102	103	104	105
Zeichen:	H	A	L	L	O	NUL
Wert:	72	65	114	114	117	0

argv[1] bis argv[argc] jeweils einen Zeiger auf Char darstellt.

Jetzt kommt der zweite Interessante Punkt: Strings – also Zeichenketten –

argv[0] die Adresse des ersten übergebenen String-Parameters enthält, argv[1] die des zweiten und so weiter. (Siehe dazu auch das Bild unten auf die-



DIE KOMPONENTEN VON ARGV zeigen auf die jeweils ersten Bytes der "Argument-Strings".



formatiert – aber immerhin: Das Programm funktioniert.

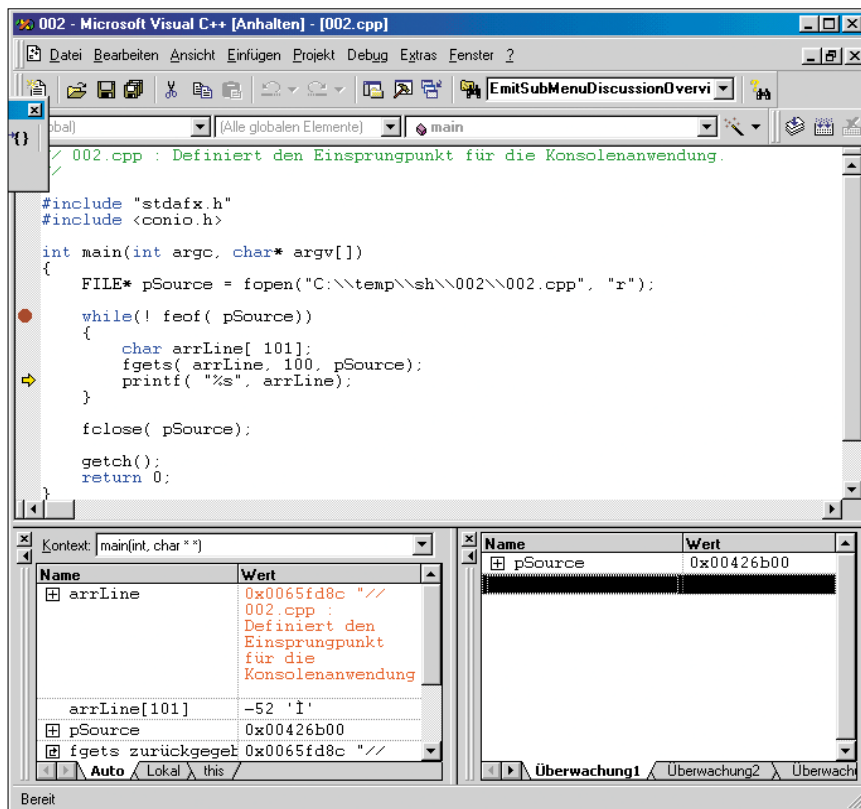
Das sollte aber wirklich das letzte Mal sein, dass Sie ein in der Entwicklung be-

nicht viel, oder, um genau zu sein, keine Datei. Das ist aber ein guter Zeitpunkt, um sich mit dem Hilfesystem von VC++ bekannt zu machen, denn die Online-

klärung für die Parameter einer Funktion benötigen.

Was Sie nun suchen, sind Funktionen, die mit Dateien zu tun haben – auf Englisch "file functions". Gehen Sie also in die *Suche* der Online-Hilfe, und geben Sie dort genau diese beiden Begriffe ein. An neuer Stelle erscheint dann ein erster Hinweis: Dort taucht eine Funktion "freopen" aus dem VC++-Programmierhandbuch auf. Diese Seite lassen Sie sich anzeigen. Die Dokumentation für freopen selbst ist zunächst nicht von Interesse – aber am Ende der einzelnen Funktionsdokumentationen gibt es immer Hinweise auf verwandte Funktionen – und wenn Sie bei freopen ans Ende scrollen, finden Sie dort einen Link zu den "Stream IO Routines". Das ist die Hilfe, die Sie suchen, denn hier finden Sie eine Übersicht über alle I/O-Funktionen, die in der C-Laufzeitbibliothek enthalten sind. Es lohnt sich, diese und ähnliche Übersichten zu studieren, denn nur so werden Sie eine Übersicht über die Ihnen zur Verfügung stehenden Funktionen erhalten. Dabei hilft übrigens auch ein Buch weiter, das sich der C-Laufzeitbibliothek widmet. Es ist dabei gar nicht so wichtig die einzelnen Funktionen zu verstehen – wichtig ist, dass man weiß, welchen Funktionsumfang es in etwa gibt, denn nur so ist man in der Lage, später auch danach zu suchen.

Wenn Sie eine Datei lesen, müssen Sie diese Datei zunächst einmal zum Lesen öffnen. Das geht mit der Funktion *fopen()* (File OPEN). *fopen()* erwartet zwei Parameter. Der erste legt die Datei



ROTE PUNKTE IM QUELLCODE-EDITOR geben an, wo sich Breakpoints befinden. Der gelbe Pfeil sagt aus, an welcher Stelle im Programm Sie sich gerade befinden.

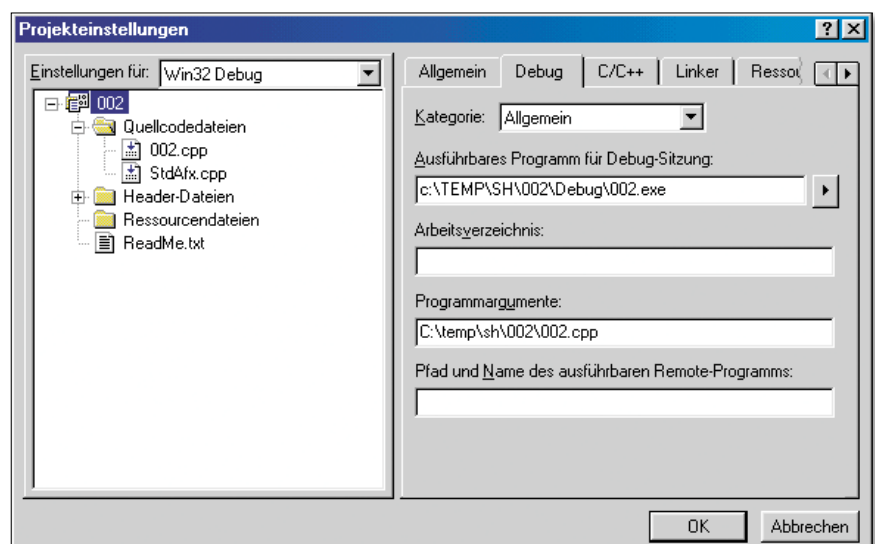
findliches Programm außerhalb des Debuggers aufrufen – denn alles, was Sie nur tun müssen, ist, dem Debugger mitzuteilen, mit welchen Parametern dieser das Programm starten soll. Das geht natürlich innerhalb der IDE und zwar im *Arbeitsbereich*-Fenster unter den *Einstellungen* zum Projekt. Diese erreichen Sie mit einem rechten Mausklick auf den Namen des Projektes im Arbeitsbereich.

Daraufhin öffnet sich ein Dialog mit mehreren Reitern, und unter dem Reiter *Debug* können Sie die gewünschten Programmargumente eingeben. Für den Anfang reicht es hier, wenn Sie ein Argument eingeben – und zwar den kompletten Pfad auf die anzuzeigende Datei. Dazu können Sie zum Beispiel den Pfad zum Quellcode Ihres Programms verwenden: Das Programm zeigt dann (später) seinen eigenen Quellcode an.

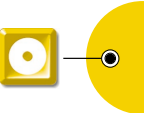
■ Dateien lesen

Sie sind nun schon ein ganzes Stück weitergekommen – doch angezeigt wird noch

Hilfe werden Sie häufig brauchen: zum Beispiel immer dann, wenn Sie nach einer Funktion suchen, deren Namen Sie nicht kennen, oder wenn Sie eine Er-



UNTER DEN PROJEKTEINSTELLUNGEN können Sie auch angeben, welche Parameter an Ihr Programm innerhalb des Debuggers übergeben werden sollen.



fest, die geöffnet werden soll, und der zweite bestimmt den Modus. Sie können Dateien nämlich für verschiedene Tätigkeiten öffnen – zum Beispiel um in die Datei hineinzuschreiben oder um aus der Datei zu lesen, und diese Modi unterscheiden sich logischerweise.

Die Datei, die angezeigt werden soll, wird Ihnen als `argv[1]` übergeben – das haben Sie bereits herausgefunden. `fopen()` müssen Sie dann folgendermaßen aufrufen:

```
fopen( argv[1], "r");
```

Das "r" steht dabei für "read" – also lesen, denn das ist der Modus, in dem die Datei geöffnet werden soll.

Das Öffnen allein reicht aber noch nicht aus: Zwar kann aus der Datei gelesen werden, doch die Funktionen zum Lesen wollen später ebenfalls wissen, welche Datei gemeint ist. Dazu liefert `fopen()` einen Rückgabewert – und zwar einen Pointer to FILE. FILE ist dabei eine Datenstruktur, die eine Datei beschreibt – die Interna von FILE sind dabei völlig irrelevant. Wichtig ist nur, dass Sie sich diesen Zeiger merken, denn den brauchen Sie bei späteren Operationen mit der Datei. Der komplette Aufruf lautet also nun:

```
FILE* pFile =  
    fopen( argv[1], "r");
```

Sie merken sich also den Rückgabewert von `fopen()` in einer Variable namens `pFile` (Pointer to File), vom Typ `FILE*`. Damit können Sie nun weiterarbeiten.

Der einfachste Weg Zeilen aus einer Textdatei zu lesen, ist die Funktion `fgets()`. Diese liest eine Zeile aus einer Datei, wobei die maximale Anzahl der zu lesenden Zeichen angegeben werden kann. `fgets()` erwartet 3 Parameter: einen Zeiger auf einen Speicherbereich, in dem die gelesene Zeile abgelegt werden soll, die maximale Anzahl an Zeichen, die gelesen werden sollen, und einen Zeiger auf FILE, der die auszulesende Datei bestimmt.

Fürs erste soll es ausreichen, wenn Sie eine einzelne Zeile aus der Datei lesen und anzeigen. Dafür brauchen Sie zunächst einmal einen Speicherbereich, in dem Sie diese Zeile ablegen können. Der Einfachheit halber geschieht dies zunächst so, dass Sie maximal 100 Zeichen der Datei auslesen – das bedeutet, Sie brauchen ein character-Array der Länge 101. (Nicht vergessen: Am Ende eines Strings steht immer ein NUL-Zeichen, und das braucht natürlich auch Platz. Daher muss Ihr Array ein Zeichen größer sein, als die Anzahl der Zeichen,

die Sie lesen möchten, denn ans Ende platzieren die C-Funktionen immer diesen NUL-Wert.)

Sie schreiben also Folgendes auf:

```
char arrLine[ 101];
```

und haben damit 101 Bytes für Characters reserviert. Dann lesen Sie mit Hilfe von `fgets()` 100 Zeichen aus der zuvor geöffneten Datei,

```
fgets( arrLine, 100, pSource);
```

und geben diese gelesenen Zeichen schließlich am Bildschirm aus:

```
printf( arrLine);
```

■ Dateien schließen

Zu Beginn Ihres Programms öffnen Sie die auszulesende Datei mit der Funktion `fopen()`. Nachdem man Dateien vor der Arbeit damit öffnen muss, kann man sich leicht denken, dass man die Dateien nach getaner Arbeit auch wieder schließen muss. Dieser Gedanke ist genau der richtige, und die Funktion zum Schließen einer Datei hat auch den erwarteten Namen. Am Ende Ihres Programms schließen Sie die Datei noch mit `fclose()`.

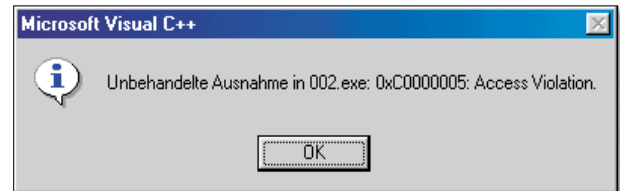
Das komplette Programm sieht nun wie in Listing 3 aus.

Wenn Sie dieses Programm übersetzen und starten ([F5]), wird die erste Zeile Ihres C++-Quellcodes am Bildschirm angezeigt. Es stellt sich nun die Frage, warum Sie der Funktion `fgets()` einfach den Namen Ihres Arrays als ersten Parameter übergeben können – schließlich erwartet die Funktion, wie man der Online-Hilfe leicht entnehmen kann, einen Zeiger auf Char, und nicht etwa ein Array aus Char. Der Grund dafür ist einfach: Zeiger auf Char – also

Strings – werden andauernd benötigt, und daher kann man in C/C++ an Stellen, an denen ein Zeiger auf Char erwartet wird, auch einfach den Namen eines Character-Arrays übergeben. Tut man dies, so wird automatisch die Adresse des ersten Zeichens im Array ermittelt und verwendet – und das ist genau der Pointer to Character, der benötigt wird. (Es gibt auch eine andere Methode, an diese Adresse zu gelangen, doch dazu später mehr.)

■ Der erste Absturz

Während der Entwicklungsphase eines Programms kommt es regelmäßig zu Abstürzen. Das ist eine ganz hervorragende und hilfreiche Eigenschaft von unfertigen Programmen, denn nur in ei-



OOPS - I DID IT AGAIN: Diese Meldung werden Sie häufiger zu sehen bekommen und zwar immer dann, wenn ein Null-Pointer-Fehler aufgetreten ist.

nem Programm, das abstürzt, kann man Fehler finden und beseitigen. Je mehr Programmabstürze in der Entwicklungsphase beseitigt wurden, umso weniger werden später beim Anwender auftreten. Mit anderen Worten: Programmabstürze kann man gar nicht genug haben und wenn man diese mit Gewalt erzwingen muss. Genau das passiert im nächsten Teil dieses Beitrages.

Das gemeinsam entwickelte Programm hat mittlerweile bereits (mindestens) einen Fehler und der muss beseitigt werden, bevor das Programm den Anwendern angeboten wird. (Eigentlich ist es erstaunlich, dass man in der Lage ist, bereits in einem gerade mal acht

Zeilen umfassenden Programm, einen Fehler einzubauen – aber vertrauen Sie an dieser Stelle dem Autor: Man kann in C/C++ noch deutlich mehr als nur einen Fehler in acht Zeilen unterbringen !)

Öffnen Sie nun den Dialog zur Einstellung der Projekteigenschaften und zwar

LISTING 3

```
int main( int argc, char* argv[])  
{  
    FILE* pSource = fopen( argv[1], "r");  
    char arrLine[101];  
    fgets( arrLine, 100, pSource);  
    printf( arrLine);  
    fclose( pSource);  
    getch();  
    return 0;  
}
```

an der Stelle, an der Sie die Programmmargumente eingetragen haben. Dort finden Sie momentan einen Pfad, der die anzuzeigende Datei spezifiziert. Ändern Sie diese Angabe nun so, dass sie einen nicht existierenden Dateispezifiziert, also zum Beispiel:

`C:\faselwurst.txt`

Wenn Sie nun Ihr Programm erneut im Debugger starten passiert Folgendes: Zunächst sieht alles ganz normal aus,

halb so, damit Sie als Programmierer in der Lage sind, Fehler besser feststellen zu können. Immer wenn diese Adresse zum Lesen oder zum Schreiben verwendet werden soll, kann der Debugger das feststellen und Sie in Form einer Dialogbox warnen: Irgendetwas hat nicht funktioniert.

Dankbarerweise ist das Problem in solchen Fällen aber schnell lokalisiert, und das geht so: Schließen Sie die Dia-

bugger kann schnell herausgefunden werden, was hier nicht passt. Klicken Sie dazu in das *Überwachungs*-Fenster. Hier können Sie in jeder Zeile je einen Ausdruck – meist den Namen einer Variablen – eingeben, und sich dann vom Debugger den aktuellen Wert anzeigen lassen. Nun ist es bei der *fgets()*-Funktion so, dass diese drei Parameter erhält: Einer ist der Zeiger auf das Array, in dem die Zeile abgelegt werden soll, einer gibt die Länge an, und einer spezifiziert die Datei, aus der gelesen werden soll.

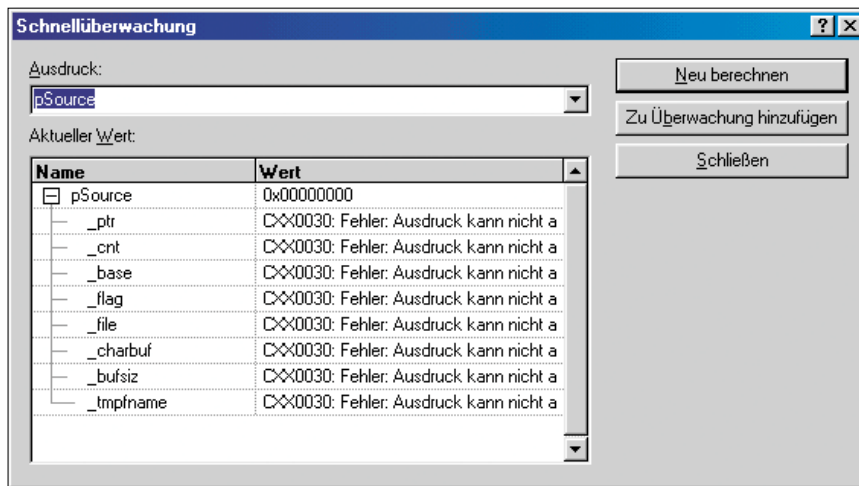
Der konstante Wert 100 wird das Problem vermutlich nicht verursachen, und auch der Name des Arrays kann eigentlich nicht der Grund sein: Schließlich hat *fgets()* damit ja noch gar nichts getan. Sie tippen also zunächst einmal den Namen der *FILE**-Variablen im Überwachungsfenster ein – also "pSource".

Der Debugger zeigt Ihnen, dass diese Variable den Wert 0x000000 – also die Adresse "0" enthält. Ein deutliches Zeichen dafür, dass irgendetwas nicht in Ordnung ist – denn Zeiger-Variable, mit denen gearbeitet wird, dürfen nie diesen Wert annehmen, es sei denn, das wird aus anderen Gründen ausdrücklich erwartet. Wie kommt es also nun dazu, dass "pSource" auf "0" zeigt? – Auch das ist mit dem Debugger schnell herausgefunden. Dazu müssen Sie aber noch weitere Elemente des Debugger erlernen, und zwar "Breakpoints" (Haltepunkte). Beenden Sie dazu zunächst Ihre Debug-Sitzung (im *Debug*-Menü gibt's den passenden Befehl).

Haltepunkte platzieren Sie innerhalb des Quelltext-Editors mit dem entsprechenden Menü-Befehl, oder mit der Taste *[F9]*. Wenn Sie das erste Mal in einer Zeile *[F9]* drücken erscheint links am Rand ein roter Kreis, mit einem zweiten Druck auf diese Taste wird

der Kreis wieder entfernt. Ist der Kreis sichtbar, ist in dieser Zeile ein Breakpoint definiert. Und das bedeutet, dass der Debugger Ihr Programm an dieser Stelle anhält, und Ihnen die Möglichkeit gibt, den Debugger zu benutzen.

Mit anderen Worten: Ein Breakpoint tut genau das, was die Fehlermeldung zuvor tat – nämlich das Programm in seinem Verlauf anzuhalten – aber eben an



IM DEBUGGER können Sie sich den Inhalt von Variablen auf verschiedene Arten anzeigen lassen.

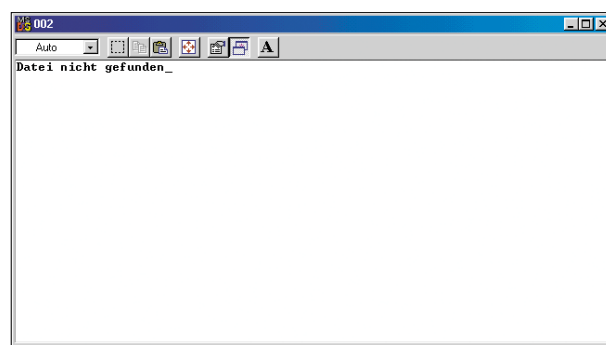
doch dann erscheint in der IDE eine Fehlermeldung mit dem Text: "Unbehandelte Ausnahme in 002.exe", gefolgt von einer Adresse in Hexadezimal-Schreibweise und der Bemerkung "Access Violation".

Geben Sie sich keine große Mühe beim Memorieren dieser Meldung – denn die werden Sie noch oft genug sehen. Stellt sich die Frage: Was ist schief gelaufen, und was will einem die IDE mit dieser Meldung sagen? Zunächst einmal bedeutet "Access Violation", genau das, was man bei einer wörtlichen Übersetzung vermuten würde: Eine Zugriffsverletzung ist aufgetreten. Eine solche Zugriffsverletzung tritt immer dann auf, wenn ein Programm versucht in einen Speicherbereich zu schreiben – oder aus einem Speicherbereich zu lesen – in den es entweder nicht lesen oder nicht schreiben darf. Sie kennen genau diese Meldung vermutlich von der Arbeit unter Windows – dort ist zwar die Dialogbox etwas hübscher, aber auch dort finden Sie bei Programmabstürzen immer wieder den Text "Zugriffsverletzung".

Ein Speicherbereich, aus dem Sie nicht lesen und in den Sie nicht schreiben dürfen, ist der Speicherbereich an und um die Adresse 0. Das ist einfach nur des-

halb so, damit Sie als Programmierer in der Lage sind, Fehler besser feststellen zu können. Immer wenn diese Adresse zum Lesen oder zum Schreiben verwendet werden soll, kann der Debugger das feststellen und Sie in Form einer Dialogbox warnen: Irgendetwas hat nicht funktioniert.

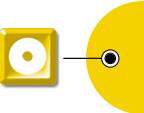
Dankbarerweise ist das Problem in solchen Fällen aber schnell lokalisiert, und das geht so: Schließen Sie die Dia-



EINE ENTSPRECHENDE FEHLERMELDUNG ist auf jeden Fall besser, als ein unerwarteter Programmabsturz.

aber vom Debugger an der Stelle gehalten, an der dieser den Fehler entdeckt hat. Die Stelle, an der sich Ihr Programm gerade befindet, ist dabei im Quellcode-Editor mit einem gelben Pfeil markiert: Das ist die Programmzeile, die als letztes ausgeführt wurde, bevor der Fehler auftrat.

In diesem Fall handelt es sich dabei um die *fgets()*-Zeile. Mit Hilfe des De-



einer vor Ihnen spezifizierten Stelle. Von Interesse ist nun der Grund dafür, warum "pSource" auf die Adresse 0 zeigt – und das lässt sich am einfachsten herausfinden, wenn Sie an der ersten Nutzung dieser Variable im Quelltext einen Breakpoint setzen: Also in der Zeile mit dem Statement:

```
FILE* pSource =  
fopen( argv[1], "r");
```

Wenn Sie den Breakpoint gesetzt haben, starten Sie das Programm erneut. (*[F5]*). Der Debugger lädt und startet das Programm, platziert es aber mit dem

vor Sie damit weiterarbeiten. Das gilt im Besonderen für Funktionen, die fehlschlagen können – so wie das bei *fopen()* der Fall ist. Die einfachste Art der Fehlerbehandlung ist es dabei, eine entsprechende Meldung auszugeben und das Programm abzubrechen:

```
FILE* pSource =  
fopen( argv[1], "r");  
if( ! pSource)  
{  
    printf("Datei  
nicht gefunden");  
    return 1;  
}
```

```
002  
Auto  
// 002.cpp : Definiert den Einsprungpunkt f"r die Konsolenanwendung.  
//  
#include "stdafx.h"  
#include <conio.h>  
  
int main(int argc, char* argv[])  
{  
    FILE* pSource = fopen("C:\\temp\\sh\\002\\002.cpp", "r");  
    while(! feof( pSource))  
    {  
        char arrLine[ 101];  
        fgets( arrLine, 100, pSource);  
        printf( "%s", arrLine);  
    }  
    fclose( pSource);  
    getch();  
    return 0;  
}
```

WENN SIE BEIM PFAD den Pfad auf den eigenen Quellcode angeben, zeigt Ihr Programm sich selbst an.

nun bereits bekannten gelben Pfeil auf der *fopen()*-Zeile. Diese Zeile können Sie nun ausführen, indem Sie die Taste *[F10]* drücken. Das führt dazu, dass genau eine Quellcodezeile ausgeführt wird, und das Programm danach wieder angehalten wird. (Um ein Programm normal weiterlaufen zu lassen, wenn es vom Debugger angehalten wurde, drücken Sie *[F5]*). Wenn Sie nun wieder den Wert von "pSource" im Überwachungsfenster überprüfen, können Sie feststellen: *fopen()* hat diesen Null-Pointer geliefert – und das ist eine Eigenschaft von vielen Funktionen in C/C++: Funktionen, die einen Zeiger liefern, liefern meist einen Null-Zeiger, wenn sie ihre Funktion nicht ausführen können. Dass *fopen()* nicht funktionieren konnte, ist ja eigentlich klar – schließlich haben Sie ja mit voller Absicht den Pfad zu einer nicht existierenden Datei angegeben.

Rückgabewerte abfangen

Es ist also wichtig, immer die Rückgabewerte von Funktionen zu testen, be-

Hier sehen Sie zum ersten Mal den *!*-Operator. Dieser Operator negiert seinen Operanden – das heißt, die Auswertung eines bool'schen Ausdrucks liefert das logisch entgegengesetzte Ergebnis zum Wert des Operanden. In diesem Fall ist der Operand die Variable "pSource".

Weiter vorne im Sonderheft haben Sie erfahren, dass der *if*-Befehl in C/C++ bool'sche Ausdrücke auswertet – und das passiert auch hier. Zwar ist die Variable "pSource" kein bool'scher Ausdruck, allerdings kann der Inhalt dieser Variable als Integer-Wert betrachtet werden, und solche können wiederum in 0 und alles andere unterteilt werden. Enthält sie die Adresse 0 (wie hier geschehen), wird der bool-

sche Ausdruck (*pSource*) zu "false" – enthält die Variable irgendeine andere Adresse, so wird der Ausdruck zu "true".

Von Interesse ist es in diesem Fall aber nur, wenn *fopen()* fehlschlägt und einen Null-Pointer liefert: Dann ist der bool'sche Wert von *pSource* "false". Zusammen mit dem Operanden *!* wird der Ausdruck (*! pSource*) also dann "true", wenn "pSource" alleine als "false" ausgewertet wird: Und das ist genau die Bedingung, die abgefangen werden muss.

Fazit

Nachdem der Fehler korrigiert ist, ist das Programm aber noch nicht komplett: Schließlich zeigt es bisher nur eine einzelne Zeile an, und nicht die komplette Datei. Was es braucht, ist offensichtlich eine weitere Schleife – und zwar eine, mit der so lange aus der Datei gelesen wird, bis das Ende der Datei erreicht ist.

Wie aber stellt man das Ende einer Datei fest? – In der Online-Hilfe werden Sie schnell fündig – die gesuchte Funktion hat den schönen Namen *feof()* – (File – End Of File). Diese Funktionen liefern einen bool'schen Wert – also true oder false. True wird dann geliefert, wenn das Ende einer Datei erreicht ist, und false, falls das noch nicht der Fall ist.

LISTING 4

```
while( ! feof( pSource))  
{  
    char arrLine[ 101];  
    fgets( arrLine, 100, pSource);  
    printf( "%s", arrLine);  
}
```

Zusammen mit einem der Schleifenkonstrukte aus dem vorigen Kapitel können Sie also die Schleife programmieren, die Sie in Listing 4 finden.

In diesem Beitrag haben Sie nicht nur Ihr erstes echtes C/C++-Programm programmiert, sondern auch Ihren ersten eigenen Absturz korrigiert: Glückwunsch – denn damit sind Sie nun in der Lage, die noch kommenden Abstürze schnell und effizient zu finden und zu beseitigen. Außerdem haben Sie zum ersten Male Zeiger-Variablen verwendet: Das werden Sie in Zukunft noch wesentlich öfter tun. UR