



## Menü-Erweiterung

# Suchen und ersetzen

Zeiger ohne Ende: Erweitern Sie den **Dateibetrachter**, und bauen Sie eine **Suchfunktion** in das Programm ein.

THOMAS WÖLFER

In diesem Beitrag geht es hauptsächlich darum, das Wissen über die Funktionsweise von Zeigern zu vertiefen. Daher sollten Sie zunächst das Heft aus der Hand legen und sich vor Ihren Rechner und das letzte Projekt setzen. Versuchen Sie dabei, das Programm um zwei Funktionen zu erweitern: Die eine Funktion soll einen Text in der Datei suchen und an die erste Zeile scrollen, in der der Suchtext gefunden wurde. Die zweite Funktion soll außerdem den Suchtext durch einen anderen Text ersetzen.

Noch ein kurzer Hinweis fürs Programmieren: Sie finden eine Funktion, mit der Sie einen Text von der Kommandozeile einlesen können – und das brauchen Sie für die Eingabe des Suchtextes – unter dem Namen *scanf()*.

Der Rest dieses Beitrags erläutert kurz die Beispiel-Implementierung der beiden Funktionen und geht dabei auf die wesentlichen Probleme ein.

### ■ Suchen – und scrollen

Logisch gesehen ist der Aufbau der Suchfunktion ganz einfach. Die Funktion muss folgende Arbeitsschritte durchführen:

- Sie muss den Anfang der Liste finden, damit auch alle Zeilen durchsucht werden.
- Sie muss dann nach dem Suchwort fragen.
- Dann muss die Funktion über alle Zeilen iterieren, und dabei untersuchen, ob eine Zeile das Suchwort enthält.

- Schließlich muss noch der "pFirst"-Zeiger gegebenenfalls auf die richtige Node gesetzt werden, damit auch an die Zeile gescrollt wird.

Der Prototyp der Suchfunktion passt zu denen der bisherigen Funktionen:

```
void Search( Node** ppNode );
```

Um nun den Anfang der Liste zu finden, kann einfach so lange nach Node-Vorgängern gesucht werden, bis es keinen mehr gibt. Um sich dabei die

zweite gibt an, an welcher Adresse der gelesene Wert gespeichert werden muss. Das bedeutet, es wird zunächst einmal Speicher fürs Ablegen des Suchwortes benötigt:

```
char szSearch[ 1024 ];
```

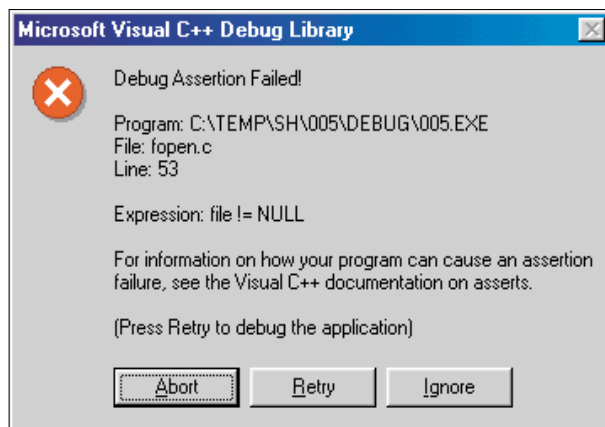
Dann kann das Suchwort eingelesen werden:

```
scanf( "%s", &szSearch[0] );
```

Die Formatangabe "%s" gibt an, dass *scanf* einen String (s) einlesen soll. Mit *scanf* können Sie aber auch beliebige andere Dinge von der Konsole einlesen – soll es zum Beispiel ein Integer-Wert sein, wäre die Formatangabe dazu "%d". Mit dem Address-Of-Operator wird die Adresse des ersten Zeichens von *szSearch* ermittelt: Ab dieser Adresse soll *scanf()* die eingelesenen Werte ablegen. Das macht auch gleich ein Problem klar: Wird ein Suchwort eingegeben, das länger als 1024 Zeichen ist, überschreibt *scanf()* wahllos Speicher. Für dieses Beispielprogramm kann man diese Problematik getrost ignorieren – Sie sollten aber im Kopf behalten, dass hier ein klarer Fehler im Programm vorliegt, der sich allerdings nur dann bemerkbar macht, wenn ein sehr langes Suchwort eingegeben wird.

Es gibt auch noch ein zweites Problem mit *scanf()*: Diese Funktion verwendet andere interne Puffer als die Funktion *getch()*, die bisher verwendet wurde. Das bedeutet im Wesentlichen, dass Sie beim Auslesen des Suchwortes mit *scanf()* nicht nur das Suchwort, sondern auch das zuletzt eingegebene Menü-Kommando erhalten: Der echte Suchbegriff beginnt also erst bei *szSearch[1]*, und nicht bei *szSearch[0]*. In der Praxis würde man dies natürlich anders lösen – aber für dieses Beispiel soll der Hinweis auf die Problematik der unterschiedlichen Puffer ausreichen.

Nun muss über alle Zeilen iteriert und das Suchwort gesucht werden. An dieser Stelle wäre es eine gute Übung, wenn



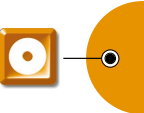
**AUCH WENN DIE FEHLERMELDUNGEN** der RTL Probleme im Quellcode von Microsoft bemängeln, bedeutet das nicht, dass dort der Fehler liegt: Vielmehr ist es so, dass dort ein Fehler von ihnen zuerst erkannt wurde.

Schreibarbeit zu erleichtern, ermittelt man zunächst aus dem Zeiger auf einen Zeiger auf einen Knoten den tatsächlichen Knotenzeiger. Mit dem wird dann iteriert:

```
Node* pN = *ppNode;
```

```
while( pN->pPrev  
       pN = pN->pPrev;
```

Dann muss das Suchwort erfragt werden. Dazu kann zum Beispiel die schon angesprochene Funktion *scanf()* verwendet werden. Diese Funktion erwartet eine Reihe von Parametern, davon sind aber zunächst nur zwei relevant: Der erste Parameter bestimmt, was eingelesen werden soll, der



Sie die Suche eines Textes innerhalb eines anderen Textes einmal selbst ausprobieren würden. Im Beispiel wurde das aber nicht gemacht – stattdessen wird die Funktion `strstr()` verwendet, die die C-RTL (RunTime Library) zur Verfügung stellt. (Genau eine solche Funktion sollten sie von Hand nachprogrammieren.)

`strstr()` erhält zwei Parameter: den Text, in dem gesucht werden soll und den Text, der gesucht werden soll. Wird der Suchtext gefunden, dann liefert `strstr()` einen Zeiger auf das erste Zeichen des Suchtextes im durchsuchten Text, ansonsten Null.

Die Iteration und die Suche kann wie in Listing 1 ausformuliert werden.

Das war doch gar nicht so schwierig!

```

\\GON\CV...VCr\Src\Dbgdel.cpp
void operator delete(
    void *pUserData
)
{
    _CrtMemBlockHeader * pHead;
    if (pUserData == NULL)
        return;
    _mlock(_HEAP_LOCK); /* block other threads */
    /* get a pointer to memory block header */
    pHead = pHdr(pUserData);
    /* verify block type */
    _ASSERT(_BLOCK_TYPE_IS_VALID(pHead->nBlockUse));
    _free_dbg( pUserData, pHead->nBlockUse );
    _munlock(_HEAP_LOCK); /* release other threads */
    return;
}
#endif /* _DEBUG */
    
```

**WIRD EIN FEHLER ENTDECKT**, platziert ihn der Debugger auf der Quellcode-Zeile, in der er entdeckt wurde: Auch, wenn es sich dabei um den Quellcode der C-Laufzeitbibliothek handelt, ist davon auszugehen, dass der Fehler im eigenen Quellcode ausgelöst wurde.

### ■ Suchen und ersetzen

Beim Ersetzen eines Suchwortes durch einen anderen Text kommen neue Probleme hinzu: Wenn das neue Wort länger ist als das alte, wird natürlich auch die Zeile länger. Es steht aber nur genau so viel Speicher zur

alte Speicher freigegeben und neuer Speicher alloziert werden muss. So etwas ist immer mit Fehlern verbunden: Mal gibt man den falschen Speicher frei, dann überschreibt man Speicher, den man eigentlich nicht überschreiben will – und schließlich ist die Berechnung der benötigten Speichergröße immer eine Quelle von Fehlern: Verrechnet man sich dabei, wird irgendwann ganz sicher irgendetwas überschrieben, das besser nicht überschrieben worden wäre.

Diese Tatsachen sind aber ein altbekanntes Problem, und die C-RTL nimmt sich dieser Problematik auch an. In der Debug-Version dieser Library können bestimmte Speichertest-Funktionen aktiviert werden: die Überprüfen, ob Speicher korrekt alloziert, freigegeben und benutzt wird. Ist das nicht der Fall, werden entsprechende Fehlermeldungen ausgegeben.

Einige grundlegende Funktionen beim Speichertest sind in Debug-Builds immer eingeschaltet, aber man kann auch zusätzliche aktivieren. Das geht mit der Funktion `_CrtSetDbgFlag()`, die im Beispielcode zu Beginn der `main()`-Funktion, wie in Listing 2 aufgerufen wird.

#### LISTING 1

```

while( pN->pNext)
{
    if( strstr( pN->pLine, szSearch+1))
    {
        // gefunden !
        (*pNode) = pN;
        return;
    }
    pN = pN->pNext;
}
    
```

#### LISTING 2

```

#ifdef _DEBUG
    _CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF |
        _CRTDBG_CHECK_ALWAYS_DF );
#endif
    
```

Verfügung, wie die Zeile lang ist. Das bedeutet, dass im Falle einer Ersetzung der

Was die Parameter für `CrtSetDbgFlag()` tun, können Sie leicht in der Online-Hilfe nachlesen. Interessanter ist das `#ifdef/#endif`-Statement um den

### VC++ UND DIE PORTABILITÄT

VC++ steht nicht nur in Versionen für das ganz normale Windows zur Verfügung: Es existieren auch andere Versionen, so zum Beispiel eine für Windows CE, das Betriebssystem für Pocket PCs. Trotzdem kann man bei VC++ nicht wirklich von einer Entwicklungsumgebung sprechen, die auf die Herstellung portabler Programme spezialisiert ist.

Wenn man lieber Programme schreiben möchte die für verschiedene Betriebssysteme übersetzt werden können, dann bietet sich der GNU C/C++ Compiler an. Dieser Compiler steht auf praktisch jeder Betriebssystemplattform zur Verfügung. Egal ob 32 oder 64 Bit Linux, und egal ob Windows oder OS/2: Eine Version des GNU ist

immer vorhanden. Obendrein liegt der GNU noch im Open Source Verfahren vor und ist damit kostenlos.

Allerdings sollte man sich nicht allzu große Hoffnungen machen, wirklich aufwendige Programme portabel mit C++ zu entwickeln, auch dann nicht, wenn man ein überall zur Verfügung stehendes Werkzeug wie den GNU C++ Compiler verwendet. Der Grund dafür ist einfach der, das mit wenigen Ausnahmen praktisch jedes etwas umfangreichere Programm auch Funktionen des Betriebssystems nutzen muss. Muss ein Fenster angezeigt oder ein Menü erzeugt werden – immer sind es Betriebssystemfunktionen die verwendet werden, und die sind nun mal per Definition nicht por-

tabel. Verwendet ein Programm solche Funktionen, dann ist es das auch nicht.

Aber auch für diese Problematik gibt es Lösungen. Die liegen fast alle in Form von kommerziellen Klassenbibliotheken vor. Diese Bibliotheken kapseln dabei Betriebssystemfunktion wie die Funktion zum Öffnen eines Fensters. Wird das Programm dann für eine bestimmte Plattform übersetzt, dann kümmert sich die Version der Klassenbibliothek für genau diese Plattform darum, dass die richtige Funktion aufgerufen wird. Mit anderen Worten: Man benötigt dann für jede Zielplattform eine andere Version der Klassenbibliothek.



Funktionsaufruf herum, denn dabei handelt es sich um ein bisher noch nicht verwendetes Element. Wie bereits erwähnt, gibt es vor dem eigentlichen Kompilervorgang noch einen Präprozessor-Durchlauf. Dabei finden Textersetzungen statt, und genau das machen Sie sich hier zu Nutze. Der Präprozessor kann nämlich auch bestimmte Bedingungen prüfen, so zum Beispiel die Bedingung, ob eine bestimmtes Symbol definiert ist. Ein solches Symbol ist zum Beispiel `_DEBUG` – dieses Symbol ist bei "Debug"-Builds definiert. (Das können Sie leicht überprüfen, indem Sie die Compiler-Optionen des Projekts für Debug-Builds im Arbeitsbereich ansehen.) Mit anderen Worten: Der Funktionsaufruf von `CrtSetDbgFlag()` findet nur in der Debug-Variante des Programms statt, denn das `_DEBUG`-Symbol ist in der Release-Variante nicht definiert.

Nun aber noch kurz zurück zur Ersetzungsfunktion: Im Wesentlichen iteriert diese genau so über den Text, wie das auch die Suchfunktion tut. Allerdings bricht die Ersetzungsfunktion nicht nach dem ersten gefundenen Text ab, und berücksichtigt außerdem auch, dass das zu ersetzende Wort mehrfach in einer Zeile vorkommen kann. Dabei passiert im Einzelnen das Folgende:

```
while( strstr( pN->pLine,
             ↪szSearch+1))
{
```

Wenn man in diesen Codeblock gelangt, wurde das Suchwort in der momentan untersuchten Zeile gefunden. Nun müssen verschiedene Längen-Berechnungen stattfinden. (Bitte beachten Sie, dass der Quellcode nicht gerade der effizienteste ist – der Grund dafür ist der, dass das generelle Vorgehen dargestellt werden soll, und nicht der "perfekt" richtige Code für die gestellte Aufgabe.)

Von Interesse sind die Längen des Suchwortes, des Ersatzwortes, die Länge der alten und die Länge der neuen Zeile, wie Sie in Listing 3 sehen.

Ferner ist von Interesse, wo innerhalb der Zeile das Suchwort beginnt, denn mit dieser Information kann die Länge des Text-

tes bis zum Suchwort ermittelt werden (s. Listing 4).

An dieser Stelle sollte Ihnen etwas auffallen: Hier wird eine Zeiger-Variable von

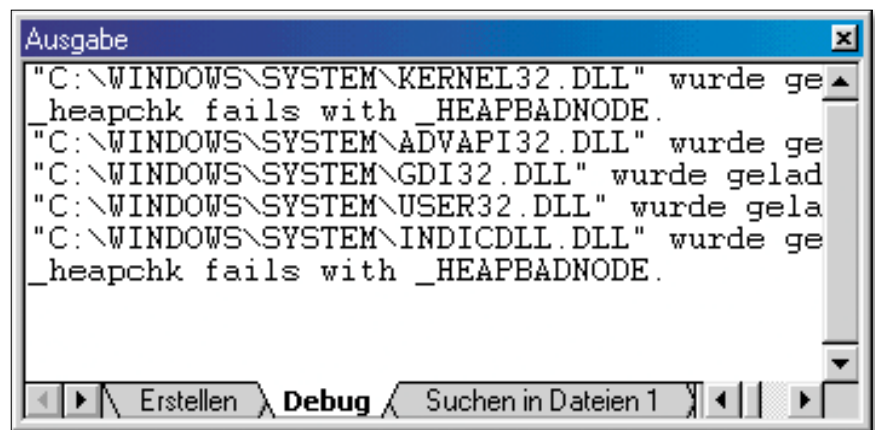
### LISTING 4

```
char* pBegin = strstr(pN->pLine,
                     ↪szSearch+1);
int cbL = pBegin - pN->pLine;
```

```
if( cbL <= cbNew)
    pNew[ cbL] = 0;
```

Schließlich kann das Ersatzwort hinten an den string gehängt werden und zuletzt passiert noch das Gleiche mit dem Rest des Textes aus der Zeile – natürlich erst ab dem Ende des Suchwortes:

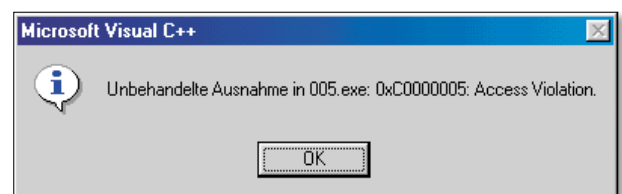
```
strcat( pNew, szReplace);
strcat( pNew, pBegin+
       ↪cbSearch);
```



**WENN SIE DIE SPEICHERTESTFUNKTIONEN AKTIVIEREN**, erscheint im Fehlerfall eine entsprechende Meldung im "Debug"-Fenster der "Ausgabe".

einer anderen abgezogen: Berechnungen mit Zeiger sind nämlich auch möglich. `pBegin` zeigt auf den gesuchten Text in der Zeile, der Zeiger zeigt also auf eine Adresse, die größer ist, als die Adresse der Zeile selbst. Zieht man nun von der einen Adresse die andere ab, erhält man die Differenz – und das ist ein Integer-Wert, der genau der Länge vom Anfang der Zeile bis zum gesuchten Wort entspricht. Nun kann der neue Speicherbereich in der richtigen Größe angefordert werden, und der erste Teil des alten Textes kann in den neuen Block kopiert werden (s. Listing 5). Dabei müssen Sie berücksichtigen (siehe Online-Doku-

Zu guter Letzt muss noch der alte Speicherblock freigegeben werden, und der Zeiger für die aktuelle Zeile auf den neu-



**DIESE MELDUNG** dürfte Ihnen mittlerweile bekannt vorkommen: Irgendwo ist wohl noch ein Fehler im Programm.

en Block mit dem ersetzten Text gebogen werden. Fertig ist die Ersetzungsfunktion:

```
delete []pN->pLine;
pN->pLine = pNew;
```

Mit dem bisher angesammelten Grundlagenwissen sind Sie schon recht tief in der Materie eingedrungen: Daher verlassen wir im Heft nun die eher "C"-artigen

Wege und wechseln mehr in den Bereich von C++.

Im nächsten Beitrag bauen Sie nochmals einen Datei-Viewer, verwenden dazu aber das Klassen-Konzept von C++.

UR

### LISTING 3

```
int cbSearch = strlen( szSearch+1);
int cbReplace = strlen( szReplace);
int cbLine = strlen(pN->pLine);
int cbNew = cbLine - cbSearch + cbReplace;
```

mentation), dass die Funktion `strcpy()` nicht immer einen Null-Terminator an den kopierten String hängt. Das muss also per Hand erledigt werden:

### LISTING 5

```
char* pNew = new char[ cbNew + 1];
strcpy( pNew, pN->pLine, cbL);
```