Fremdgänger

Borlands IDE ruft externe Compiler auf

Stefan Kuhlins

Lehrstuhl für Wirtschaftsinformatik III
Universität Mannheim
68131 Mannheim
stefan@kuhlins.de

3. Mai 1993

Zusammenfassung

Die integrierte Entwicklungsumgebung von Borland C++ für DOS ist wegen ihrer hohen Funktionalität sehr beliebt bei Programmierern. Mit etwas Know-how lassen sich beliebige Programme von ihr aus aufrufen, zum Beispiel Clipper.

Am Beispiel von *grep*, das standardmäßig installiert ist, bekommt man ein Gefühl für die Arbeitsweise beim Aufruf externer Programme aus der IDE [1]. Nachdem der Benutzer den Menüpunkt *grep* aus dem Systemmenü von Borland C++ angewählt hat, wird das Programm gestartet. Es leitet seine Ausgabe wie gehabt nach *stdout* beziehungsweise *stderr* weiter. Dort wird sie via *stdin* von einem kleinen Filterprogramm übernommen. Dieses Programm übersetzt die *grep*-Ausgabe in das Format, das die IDE versteht, und gibt sie abermals über *stdout* weiter.

Die durch *grep* gefundenen Sourcecode-Stellen werden im Message-Fenster der IDE angezeigt. Sie können per Mausklick in den Editor geladen werden, genau so, wie man es von den Fehlermeldungen des eingebauten Compilers her gewohnt ist. Damit ist klar, was zu tun ist, um die Sprachprobleme der IDE zu beseitigen: Entwickeln und Installieren eines Filterprogrammes.

Das Filterprogramm (siehe Anhang A) ist in C++ geschrieben [2] und wurde zusammen mit der Clipper-Version 5.01a sowie Borland C++ 3.1 getestet. Für die alte Clipper-Version vom Sommer ’87 sind einige Anpassungen notwendig (siehe Anhang B). Das Filterprogramm extrahiert aus der Textausgabe des Compilers die Namen der kompilierten Dateien, den Meldungstext sowie die Nummern der Zeilen und – falls vorhanden – die der Spalten. Diese Daten werden im passenden Format (siehe Abb. 1) an die IDE weitergereicht.

```
ID
newFile <Dateiname>
   newLine <Zeilennummer> <Spaltennummer> <Meldungstext>
   newLine <Zeilennummer> <Spaltennummer> <Meldungstext>
   ...
newFile <Dateiname>
   ...
...
endID
```

`ID`, `endID`, `newFile` und `newLine` werden als Konstanten am Anfang des Programmes definiert (siehe Anhang A). `<Dateiname>` und `<Meldungstext>` sind Strings, die mit einer Null enden müssen. `<Zeilennummer>` und `<Spaltennummer>` müssen als 2 Byte Integerwerte binär übergeben werden.

Abbildung 1: Datenformat für das Message-Fenster der IDE

Die Installation des Compilers und des Filterprogrammes erfolgt im Transfer-Dialogfenster der IDE. Für das Beispiel können die Angaben folgendermaßen aussehen:

```
Program Title:
  -Clipper
Program Path:
  C:\CLIPPER5\BIN\CLIPPER
Command Line:
  $EDNAME -q -m -s -w
  $NOSWAP $SAVE CUR
  $CAP MSG(CLIP2MSG)
```

Das Tilde-Zeichen (`~`) im Programmnamen (`Program Title`) markiert den Hot Key, mit dem der Menüpunkt verbunden ist. Die mit einem `$`-Zeichen beginnenden Angaben, von Borland Makros genannt, werden von der IDE interpretiert. Sie legen die Einzelheiten zum Aufruf des Compilers fest und sind in Borlands Online-Dokumentation ausführlich erklärt. Nicht zu vergessen ist das Makro `$SAVE CUR`, das dafür sorgt, daß die aktuelle Datei vor dem Aufruf des Compilers abgespeichert wird. Die Angaben `-q -m -s -w` sind für den Clipper-Compiler bestimmt.

Literatur

[1] Borland C++ 3.1, Benutzerhandbuch

[2] Martin Schader und Stefan Kuhlins, Programmieren in C++, Einführung in den Sprachstandard C++ Version 3.0, Springer-Verlag, Heidelberg 1993

A Programmlisting

Das C++-Programm wandelt die Meldungen des Clipper-Compilers 5.x in das Format der Borland IDE um.

```
/* Clip2Msg - V 1.0 (C) Stefan Kuhlins 1993
 * Dieser Filter konvertiert die Warnungen und Fehlermeldungen
 * des Clipper-Compilers 5.x in das Format des Message-Fensters
 * der Borland IDE.
 */
#include <iostream.h>               // cin, cout etc.
#include <strstrea.h>               // istrstream
#include <string.h>                 // strlen(), strncmp()
#include <io.h>                     // setmode()
#include <fcntl.h>                  // O_BINARY

const char ID[]    = "BI#PIP#OK";   // Erkennung am Dateianfang
const char newFile = '\0';          // neue Datei
const char newLine = '\1';          // neue Zeile
const char endId   = '\x7f';        // Dateiende

union iorc {                        // "int oder char"
  int  i;                           // zum Einlesen int
  char c[2];                        // zur Ausgabe char*
};

const iorc col      = { 1 };        // immer erste Spalte
const char fileId[] = "Compiling";  // neue Datei bei Clipper
const int  fIdLen   = 9;            // entspricht strlen(fileId)
const int  bufsize  = 256;          // Puffergroesse, ggf. erhoehen

int main()
{
  char* inbuf    = new char[bufsize];  // Eingabepuffer,
  char* filename = new char[bufsize];  // Dateinamen inkl. Pfad
  char* errmsg   = new char[bufsize];  // Fehlermeldung
  iorc line;                           // Zeilennummer

  setmode(1, O_BINARY);  // stdout in den Binaermodus schalten
                         // sonst wird LF in CR/LF umgewandelt
  while (cin.getline(inbuf, bufsize))  // zeilenweise einlesen
    if (strncmp(inbuf, fileId, fIdLen) == 0) {  // Erste Datei?
      istrstream instr(inbuf);  // zum Extrahieren der Infos
      instr.ignore(fIdLen+1);   // fileId und ' ' ueberspringen
      instr >> filename;
      break;                    // Start-Schleife abbrechen
    }
  if (!cin.good()) return 1;    // bei Fehler abbrechen

  cout.write(ID, strlen(ID)+1);     // ID inkl. '\0' Terminator
  cout.put(newFile);                // neue Datei beginnen
  cout.write(filename, strlen(filename)+1);
```

```
  while (cin.getline(inbuf, bufsize)) {
    istrstream instr(inbuf);
    if (strncmp(inbuf, fileId, fIdLen) == 0) {  // Neue Datei?
      instr.ignore(fIdLen+1);       // s.o.
      instr >> filename;
      cout.put(newFile);            // s.o.
      cout.write(filename, strlen(filename)+1);
    }
    else if (strncmp(inbuf, filename, strlen(filename)) == 0) {
      instr.ignore(strlen(filename)+1);  // Name + ( uebergehen
      instr >> line.i;                   // Zeilennummer einlesen
      instr.ignore(1);                   // ) ueberspringen
      instr >> ws;                       // kein white space
      instr.getline(errmsg, bufsize); // Fehlermeldung einlesen

      cout.put(newLine);                 // neue Zeile anfangen
      cout.write(line.c, 2);             // Zeilennummer,
      cout.write(col.c, 2);              // Spalte und
      cout.write(errmsg, strlen(errmsg)+1); // Meldung eintragen
    }
  }
  cout.put(endId);       // Ausgabe beenden

  delete[] inbuf;        // immer sauber bleiben ...
  delete[] filename;
  delete[] errmsg;
  return 0;              // ordentliches Programmende
}
```

B Clipper-Compiler 5.x und Sommer 87

```
/*
 * Clip2Msg - V 1.0b (C) Stefan Kuhlins 1993
 *
 * Dieser Filter konvertiert die Fehlermeldungen des
 * Clipper-Compilers 5.x (mit und ohne Option -q) und
 * des alten Clipper-Compilers Sommer 87 in das Format
 * des Message-Fensters der Borland IDE.
 *
 * Beim alten 87er-Clipper ist die Bestimmung der Spalte,
 * in der der Fehler vermutet wird moeglich, wenn OHNE
 * die Option -q compiliert wird. ueber die folgende
 * #define-Anweisung wird festgelegt, ob der Filter fuer
 * die Clipper-Option -q compiliert wird oder nicht.
 */

#undef S87Q                          // Clipper S87 ohne -q
//#define S87Q                       // Clipper S87 mit -q

#include <iostream.h>                // cin, cout etc.
#include <strstrea.h>                // istrstream
#include <string.h>                  // strlen(), strncmp()
#include <io.h>                      // setmode()
#include <fcntl.h>                   // O_BINARY
#include <ctype.h>                   // isdigit()

const char ID[]     = "BI#PIP#OK";   // am Dateianfang Erkennungsstring
const char newFile  = '\0';          // Neue Datei
const char newLine  = '\1';          // Neue Zeile
const char endId    = '\x7f';        // Dateiende

union iorc {                         // "int oder char"
  int  i;                            // zum Einlesen der Zeilennummer int
  char c[2];                         // ostream.write() benoetigt char*
};

      iorc col      = { 1 };         // Cursor in die erste Spalte setzen
const char fileId[] = "Compiling";   // Erkennung fuer neue Datei bei Clipper
const int  fIdLen   = 9;             // entspricht strlen(fileId)
const int  bufsize  = 256;           // Puffergroesse fuer eine Zeile
                                     // (ggf. erhoehen)
int main()
{
  char* inbuf    = new char[bufsize]; // Speicher fuer Eingabepuffer,
  char* filename = new char[bufsize]; // Dateinamen (inkl. Pfad) und
  char* errmsg   = new char[bufsize]; // Fehlermeldung reservieren
  iorc line;                         // Zeilennummer

  setmode(1, O_BINARY);              // stdout in den Binaermodus schalten
                                     // sonst wird LF in CR/LF umgewandelt
  while (cin.getline(inbuf, bufsize))          // zeilenweise einlesen
    if (strncmp(inbuf, fileId, fIdLen) == 0) {  // erste Datei suchen
```

```
      istrstream instr(inbuf);      // zum Extrahieren der Informationen
      instr.ignore(fIdLen+1);       // fileId und ' ' ueberspringen
      instr >> filename;
      break;                        // Start-Schleife abbrechen
    }
  if (!cin.good()) return 1;        // bei Fehler abbrechen

  cout.write(ID, strlen(ID)+1);     // ID inkl. '\0' Terminator
  cout.put(newFile);                // neue Datei beginnen
  cout.write(filename, strlen(filename)+1);

  while (cin.getline(inbuf, bufsize)) {
    char* p = inbuf;                // bei Clipper 5.x ohne -q
    while (isdigit(*p)) p++;        // Zeilennummern ueberlesen
    istrstream instr(p);            // zum Extrahieren der Informationen
    if (strncmp(p, fileId, fIdLen) == 0) {  // Neue Datei? (beide Compiler)
      instr.ignore(fIdLen+1);       // s.o.
      instr >> filename;
      cout.put(newFile);            // s.o.
      cout.write(filename, strlen(filename)+1);
    }
    else if (strncmp(p, "line", 4) == 0) {   // Clipper S87
      do {
        instr.ignore(5);
        instr >> line.i;
      } while (instr.peek() == 'l');  // ohne -q "line 100line 112:" abfangen
      if (instr.peek() == ':') {      // Zeilennummer der Fehlermeldung?
        instr.ignore(1);              // : ueberspringen
        instr >> ws;                  // white space ueberspringen
        instr.getline(errmsg, bufsize);  // Fehlermeldung einlesen

#ifndef S87Q                          // Clipper S87 ohne -q
        if (cin.peek() != 'l' && cin.peek() != 'C') {
          // "line ..." oder "Compiling ..." abfangen.
          // "Vermutlich" kommt Sourcecode-Zeile gefolgt von ^-Zeile.
          cin.getline(inbuf, bufsize);  // naechste Zeile ueberlesen
          cin.getline(inbuf, bufsize);  // Position von ^ feststellen
          col.i = strstr(inbuf, "^")-inbuf+1;  // Spalte setzen
        } else col.i = 1;
#endif
        cout.put(newLine);            // neue Zeile anfangen
        cout.write(line.c, 2);        // Zeilennummer,
        cout.write(col.c, 2);         // Spalte und
        cout.write(errmsg, strlen(errmsg)+1);  // Fehlermeldung eintragen
      } else if (strncmp(p+instr.tellg(), fileId, fIdLen) == 0) {  // s.o.
        instr.ignore(fIdLen+1);       // s.o.
        instr >> filename;
        cout.put(newFile);            // s.o.
        cout.write(filename, strlen(filename)+1);
      }
    }
    else if (strncmp(p, filename, strlen(filename)) == 0) {  // Clipper 5.x
      instr.ignore(strlen(filename)+1);  // Name und ( ueberspringen
```

```
      instr >> line.i;                // Zeilennummer einlesen
      instr.ignore(1);                // ) ueberspringen
      instr >> ws;                    // white space ueberspringen
      instr.getline(errmsg, bufsize); // Fehlermeldung einlesen

      cout.put(newLine);              // neue Zeile anfangen
      cout.write(line.c, 2);          // Zeilennummer,
      cout.write(col.c, 2);           // Spalte und
      cout.write(errmsg, strlen(errmsg)+1);  // Fehlermeldung eintragen
    }
  }
  cout.put(endId);                  // Ausgabe beenden

  delete[] inbuf;                   // immer sauber bleiben ...
  delete[] errmsg;
  delete[] filename;
  return 0;                         // ordnungsgemaesses Programmende
}
```