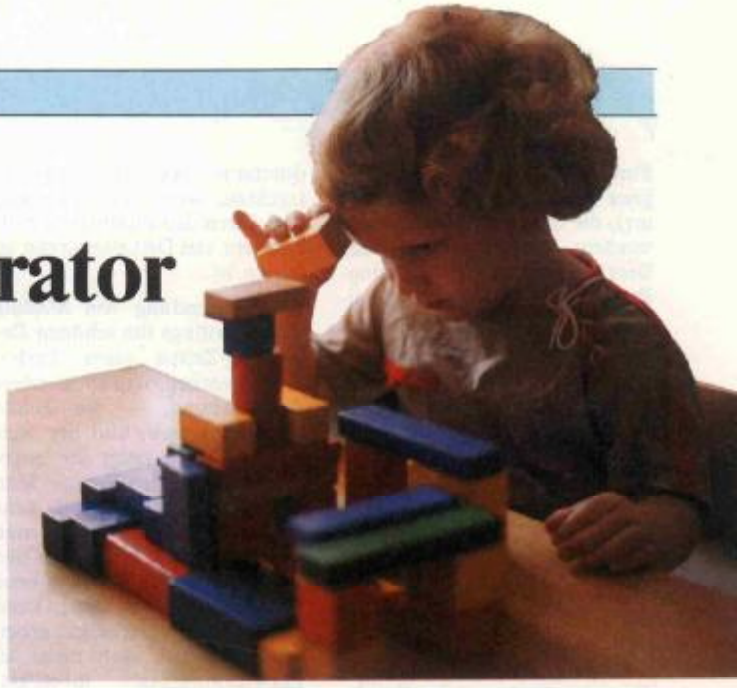
Modula-2 System für Z80 CP/M:

Der Baustein-Generator



Michael Bodag

Manchmal könnte ich meine kleine Tochter beneiden. Wenn sie sich mal wieder so richtig im Zimmer ausgebreitet hat, kann man beobachten, wie sie eines ihrer Bauwerke sinnend betrachtet, in eine der herumstehenden Spielzeugkisten greift und einen Bauklotz ans Tageslicht befördert. Dieser wird ein paarmal hin- und hergedreht und dann auf das entsprechende Gebilde gesteckt. Meist kann ich zwar ihren Ausführungen zur Konstruktion des Werkes nicht folgen, aber eines verblüfft mich immer wieder: viele kleine Klötzchen ergeben ein großes, haltbares Bauwerk. Und so wünschte ich mir denn schon des öfteren, daß es so etwas doch auch für die Konstruktion meiner Programme gäbe.

Natürlich, ich benutze Pascal, aber so ab ein paar tausend Zeilen, da verliere ich schon gelegentlich die Übersicht. Nein, nicht über die Programmstruktur, dafür strukturiert man ja. Aber diese Variablen, die da unbedingt global sein müssen... und wie hatte ich doch jenes in jener Prozedur noch gelöst...? Und daß diese Prozeduren auch immer vergessen müssen, was man gerade in ihnen getrieben hat; beim nächsten Aufruf benötige ich doch wieder genau diesen Wert! Ähnliche Gedanken müssen wohl auch den Schöpfer von Pascal geplagt haben: Er kreierte 'Modula'.

Modular

Damit ist Professor Wirth wohl ein weiterer genialer Wurf gelungen. Alle genannten Restriktionen sind in Modula-2 weitestgehend behoben. Ein Programm kann sich aus vielen Modulen (Bausteinen) zusammensetzen, und der Compiler wacht unerbittlich darüber, daß auch alle 'Anschlüsse' passen. Dies wird durch ein ausgeklügeltes System von IMPORTen und EXPORTen (von Typen, Funktionen, Prozeduren etc.) bewerkstelligt. Und nur über diese genau definierten Schnittstellen kommunizieren die Module miteinander.

Dabei ist jedem Modul vom jeweils anderen nur das bekannt, was man ausdrücklich erlaubt hat. Wie (und womit) einzelne Details im anderen Modul gelöst werden (z. B. die Implementation irgendwelcher Algorithmen) ist dem Nachbarmodul grundsätzlich völlig verborgen. Und es gibt viele Möglichkeiten, mit den Modulen zu 'jonglieren' (etwa die sogenannte Datenkapselung oder die separate Kompilation).

Alle diese Möglichkeiten auch nur anzusprechen, würde den Rahmen dieses Artikels bei weitem sprengen. Ein interessantes Detail: Modul-lokale Objekte (z. B. Variable) existieren so lange, wie die jeweilige Umgebung (das sogenannte 'environment') existiert. Womit das Problem der Vergeßlichkeit von Pascal-Prozeduren gelöst wäre.

Was nützt jedoch die schönste Sprachdefinition ohne das entsprechende Handwerkszeug, welches die Sprache mit Leben erfüllt? Auch hier kann inzwischen geholfen werden: Die Hochstrasser Computing AG in Zürich entwickelte das 'Modula-2 System for Z80 CP/M'.

Das System wurde aus der Diplomarbeit von vier Studenten der Computerwissenschaften der ETH Zürich entwickelt. Und man kann wirklich von einem System sprechen. Die Lieferung besteht aus rund 700 KByte Programmen. Da sind zunächst der Compiler (4 Pass) und der Zwei-Pass-Linker. Man kann den Compiler über 'Schalter' (Switches) unter anderem anweisen, ein M80-kompatibles MAC-File zu erstellen. Normalerweise wird ein sogenanntes MRL-File mit relokalisierbarem, systemeigenem Code produziert. Ein Lister erstellt im Falle von Kompilationsfehlern ein File mit Fehlermeldungen. Der Reference Lister dient zur Erstellung normaler Listings. Auch er läßt sich über Switches zu vielerlei Aktivitäten bewegen, inklusive der Erstellung einer Cross-Referenz-Tabelle. Ein Konverter bringt REL-Files ins nötige MRL-Format, und ein Konfigurationsprogramm bietet die Möglichkeit, die Suchwege von Compiler und Linker zu beeinflussen.

Vielfalt

Nein, das ist noch nicht alles! Die mitgelieferten Utilities kann man im Dutzend zählen (das ist wörtlich gemeint), und drei Testprogramme (Ackermann — Eratos — Eight-Queens) sind dann auch noch da.

Und das (englischsprachige) Handbuch nicht zu vergessen! Auf etwa 210 Seiten enthält es dichtgedrängt Informationen. Es ist sehr sauber gegliedert und in flottem Stil mit Humor geschrieben. Gleich am Anfang stehen etliche 'fair warnings': Das Buch versteht sich weder als Reference Manual noch als Programmierkurs in Modula-2. Es ist hauptsächlich als Unterstützung für einen Pascal-Programmierer gedacht, der sich mit den Unterschieden zu Modula-2 vertraut machen will. Dabei setzt der Autor des Handbuchs ein gerüttelt Maß an Basiswissen über CP/M und vor allen Dingen über Pascal voraus.

Syntaxbeschreibungen sind in der sogenannten EBNF-Notation gehalten (Extended Backus-Naur-Formalism). Weiterhin werden im Handbuch eingehend die Implementation der Sprache, Restriktionen, aber auch Möglichkeiten zum Einbinden eigener Assembler-Routinen behandelt.

Eingeschränkt?

Und da wir gerade bei den Restriktionen sind: Es gibt da einige, meist sind sie jedoch nicht gravierend. Zum Beispiel können Laufvariable in FOR-Schleifen nicht MAX(INTEGER) beziehungsweise MAX (CARDINAL) als Obergrenze haben. Beim Herunterzählen von Enumerationstypen darf nicht bis zum ersten Element (ORD(element) = 0) gezählt werden. Es können nicht mehr als sechzehn LOOPs, MODULEs, PROCEDUREs oder WITH-Statements geschachtelt werden. 'Priorities' sind nicht implementiert und, etwas unangenehmer, 'Version-Control' auch nicht. Aber das sind alles Dinge, mit denen man leben kann. Für mich persönlich ist eine andere Einschränkung wesentlich unerfreulicher: REALs sind im 'hidden bit'-Format mit vier Bytes implementiert. Dadurch ergibt sich zwar ein enormer Wertebereich, aber auch eine Auflösung von nur sieben signifikanten Stellen. Und das erscheint mir etwas wenig.

Grundsätzlich ist jedoch der Einbau eigener Ideen möglich. Modula ist eine 'offene' Sprache, das bedeutet, was nicht vorhanden ist, baut man sich selbst. So werden auch zwei Bibliotheken mitgeliefert: Die Standard- und die Utility-Bibliothek. Vorgegeben sind Prozeduren und Funktionen (eine

Funktion ist in Modula übrigens eine spezielle Prozedurart), die immer wieder benötigt werden: Write, WriteLn, das Bearbeiten von Files, String-Prozeduren, Mathe und so weiter. Wer damit nun nicht auskommt, kann das System ganz nach Belieben erweitern.

Erweitert

Dazu erstellt man sich ein sogenanntes Definition Module. Dies enthält die Schnittstelle(n) zur 'Umwelt'. Alles, was von anderen Modulen benutzt werden soll, EXPORTiert man. Bei der Kompilation dieses Moduls erstellt der Compiler ein MSY-File, ein Symbol-File. Dann benötigt man das 'Implementation Module', in dem der Algorithmus zur Lösung des jeweiligen Problems implementiert wird. Der Compiler erstellt daraus ein MRL-File (relokalisierbar).

Wenn man in einem anderen Modul irgend etwas aus diesem Bibliotheksbaustein benötigt, so wird das einfach IMPORTiert. Dabei findet eine strenge Kontrolle der Schnittstellen statt, 'Verknotungsfehler' haben keine Chance. Möchte man nun zum Beispiel ein anderes Terminal installieren, so bearbeitet man das Implementation Module, übersetzt es neu, und die Sache ist erledigt. Etwas diffiziler gestaltet sich die Angelegenheit, wenn man das Schnittstellenmodul ändert: Alle Module, die aus ihm importieren, müssen neu übersetzt werden.

Geschwindigkeit

Und wie schnell übersetzt der Compiler? Um das herauszufinden, wurde das Modul 'Random' mit verschiedenen Sprachen, Compilern und Computern bearbeitet (Tabelle zu Beginn von Programm 1). Der CS-2000 und die ITT-Rechner liefen mit Pascal MT+, der Osborne arbeitete mit Turbo-Pascal (also alles unter CP/M). Der CS-2000 ist ein nicht übermäßig bekannter Z80-Computer (4 MHz), der vor allem im Hinblick auf Diskettenoperationen sehr schnell ist. Die Werte für den Philips P2000C sind mit dem Modula-2-Compiler unter CP/M-80 ermittelt worden. Dabei wurde ein COM-File von rund 20 KByte Länge erzeugt. Die Kompilations- und Link-Zeiten sind als durchaus brauchbar zu betrachten, wenn man überlegt, daß durch den modularen Aufbau sehr viel Diskettenarbeit zu leisten ist.

Bei Verwendung von Modula sind allerdings die schönen Debugging-Zeiten eines Turbo-Pascal vorbei. WordStar erlebt Auferstehung — die Zeiten quadrieren sich, und der Nervenverbrauch steigt so ungefähr in der vierten Potenz. Man ist durch Turbo-Pascal ganz schön verwöhnt. Wenn man sich jedoch erst einmal an den neuen Compiler und die neue Sprache gewöhnt hat, kann man mit beiden sehr gut arbeiten. Und was man mehr an Entwicklungszeit investiert, wird durch die Programmsicherheit und die relativ leichte Wartung aufgewogen.

Fehler

Die Fehlermeldungen des Compilers sind sehr zahlreich, fast schon zuviel. Sie trafen jedoch nicht immer 'den Punkt'. Und gelegentlich schrieb der Compiler auf den Bildschirm, daß er das File 'xxxx' nicht öffnen könne. Welches File er meinte, konnte ich nicht entziffern, der Name bestand aus irgendwelchen Hieroglyphen. Ich wußte noch gar nicht, über welch enormen Zeichensatz mein Rechner verfügt.

Auch den Linker kann man mit vielerlei Switches steuern. Es gibt eigentlich nichts, was man beim Linker-Lauf nicht erfährt. Und selbst drehen kann man an fast jedem Detail: Startadresse des Programms, Start des Datenbereichs, Start des Heap.

Wenn man nun erfolgreich 'gelinkt' hat, steht ein COM-File von etwa 20 KByte Länge auf der Diskette. Das Modul 'Random' selbst ist daran mit 783 Bytes beteiligt. Und wie die Laufzeit beweist: der Code ist 'nicht von schlechten Eltern'. 'Random' ist zwar in keinster Weise optimiert (eher das Gegenteil), aber trotzdem wird sogar die bei Diskoperationen so schnelle CS-2000 geschlagen. Allerdings werden in dieser Programmversion in 'transfer' nicht so viele Parameter hin- und hergeschaufelt. Aber was bringt das schon im Verhältnis zur Zugriffsart auf die Diskette (in diesem Fall gepuffert)! Diese erstaunlich kurze Laufzeit veranlaßt mich, 'Random' auf einer P2000C doch auch mal unter Turbo-Pascal 3.0 und MSDOS 2.11 zu übersetzen (5 MHz 8088-Karte). In der Übersetzungsleistung hatte Modula keine Chance (Turbo benötigte weniger als drei Sekunden), aber in der Laufzeit schlägt sich der Compiler hervorragend: der Z80 war nur um zwei Sekunden langsamer als der 16-Bitter.

Ein anderer Linker-Lauf wurde mit dem als Benchmarktest oft für ungeeignet befundenen Sieb des Eratosthenes durchgeführt. 'Eratos' selbst bringt 265 Bytes, mit IMPORTs sind 1215 Bytes zu verzeichnen. Kompiliert wurde in 49 Sekunden, 'gelinkt' in 26 Sekunden. Die Laufzeit beträgt 20 Sekunden. Der Osborne unter Turbo-Pascal benötigt 29 Sekunden Arbeitszeit [1]. Man sieht: der erzeugte Maschinencode ist wirklich schnell.

Bei dieser Gelegenheit möchte ich ein paar grundsätzliche Worte zu solchen Compiler-Tests sagen: Bedauerlicherweise gibt es keine objektiven und

```
MODULE Random;

(*              CS - 2000  ITT 2020  ITT 3030  OSBORNE 1  P2000C      *)
(*              ---------------------------------------------------- *)
(* Compilation      30.2       94.0       63.5       ca. 6       65   *)
(* Linkzeit         26.0       97.2       38.7       -----       76   *)
(* Laufzeit         14.0       25.1       25.1        23.0       12 11*)
(*                                                                    *)
(* Die Compilationszeit gilt fuer Diskcompilation.                    *)
(* Es wurde KEINE neu formatierte Diskette benutzt, sie war schon mit *)
(* rund 400 KBytes belegt, CP/M hatte also allerlei mit der Suche     *)
(* nach freien Sektoren zu tun. Dies duerfte der Praxis entsprechen.  *)

FROM SYSTEM IMPORT
  ADR, SIZE;

FROM TERM1 IMPORT
  Write, WriteLn, WriteString;

FROM InOut IMPORT
  WriteCard;

FROM Screens IMPORT
  ClrScr, Bell;

FROM Files IMPORT
  FILE, FileState, FileStatus, Open, Create, Close, FilePos,
  InitPos, CalcPos, SetPos, WriteRec, ReadRec;

TYPE satz      = ARRAY [0..255] OF CHAR;

VAR  i:              CARDINAL;
     inwo, auswo:    FilePos;
     ifil, ofil:     FILE;
     ierr, oerr,
     inerr, outerr: FileState;

PROCEDURE transfer(i: CARDINAL);

VAR nn:       CARDINAL;
    satzaus: satz;

BEGIN
  nn := 19 - i;
  CalcPos(nn,SIZE(satzaus),inwo);
  SetPos(ifil,inwo);
  ReadRec(ifil,satzaus);
  ierr := FileStatus(ifil);
  IF (ierr = EndError) THEN RETURN; END;
  CalcPos(i,SIZE(satzaus),auswo);
  SetPos(ofil,auswo);
  WriteRec(ofil,satzaus);
  oerr := FileStatus(ofil);
  IF (oerr = StatusError) THEN RETURN; END;
  WriteString('+++ RECORD ');
  WriteCard(nn,0);
  WriteString(' successfully transferred to RECORD ');
  WriteCard(i,0);
  WriteLn;
END transfer;

BEGIN (* random *)
  ClrScr;
  WriteString('Progrom RANDOM in MODULA 2'); WriteLn;
  WriteString('--------------------------'); WriteLn;
  Bell;
  InitPos(inwo);
  InitPos(auswo);
  inerr  := Open(ifil,'RANDOM1.DAT');
  outerr := Open(ofil,'RANDOMO.DAT');
  IF ((inerr = FileOK) AND (outerr = FileOK))
  THEN
    FOR i := 0 TO 19 DO
      IF (NOT (ierr = EndError)) AND (NOT (oerr = StatusError))
      THEN
        transfer(i);
        IF (ierr = EndError) THEN
          WriteString('>>> INPUT ERROR, Progrom aborted.'); END;
        IF (oerr = StatusError) THEN
          WriteString('>>> OUTPUT ERROR, Progrom aborted.'); END;
      END;
    END;
  END;

  IF (inerr = FileOK) THEN inerr  := Close(ifil); END;
  IF (outerr= FileOK) THEN outerr := Close(ofil); END;

  Bell;
  WriteString('Progrom terminated ');
  IF ((ierr = EndError) OR (oerr = StatusError) OR
      (inerr # FileOK) OR (outerr # FileOK))
  THEN WriteString('ab'); END;
  WriteString('normally. Bye.");
  WriteLn;
END Random.
```

Programm 1 wurde zum Testen des Compilers verwendet.

verläßlichen Verfahren, wie so ein Test am besten durchzuführen ist. Man hat sich also selbst ein paar Kriterien erstellt, mit denen man austestet, wie gut (oder schlecht) der Compiler diese erfüllt, und wie einfach (oder schwierig) es ist, zum Ziel zu kommen. Man kann ein bißchen mit dem Compiler 'spielen', man bekommt ein gewisses 'Feeling' und einen ersten Einduck. Mehr aber nicht! Ob ein Compiler wirklich fehlerfrei und gut ist, zeigt erst die langjährige Arbeit damit.

```
A>
A>
A>
A>
A>mo random/v
MODULA-2 Compiler for Z80-CP/M  Version 15-03-1985
Compilation of RANDOM.MOD
P1..
279 bytes and 40 names in name table

Imported Modules:
    B:TERM1    .MSY - TERM1
    B:INOUT    .MSY - InOut
    B:SCREENS  .MSY - Screens
    B:FILES    .MSY - Files
P2..
629 bytes in name table
P3..
32 % of expression evaluation buffer used
P4..
Code size:  783 bytes
Data size:  14 bytes
```

```
A>ml random/v
MODULA-2 Linker for Z80 CP/M                    Version 28-03-1985
P1
Random                  Code :   783        Data :    14
Files                   Code :  4053        Data :    53
TERM1                   Code :   422        Data :     6
InOut                   Code :  1360        Data :    26
Screens                 Code :   283        Data :     0
FileNames               Code :  1677        Data :    25
Moves                   Code :    44        Data :     0
UpSys                   Code :   112        Data :     0
Texts                   Code :  2088        Data :    12
BeqIO                   Code :  3453        Data :     6
Strings                 Code :  1598        Data :     6
Conversions             Code :  1360        Data :    24

Initial Code Start :   100H   End :   125H   Length :   38
Code         Start :   125H   End :  4BF5H   Length : 19152
Data         Start :  4BF5H   End :  4D36H   Length :   321
Heap         Start :  4D37H   End :  E806H
Heap & Stack Space :  39631 bytes (on this System)
```

Screen-Dumps vom Compiler-Lauf (links) und vom Linker-Lauf. Viele Meldungen helfen, die Übersicht über das Gesamtprogramm zu behalten.

Fazit

'Modula-2 System for Z80 CP/M' (wer hat nur diesen Namen erfunden?) ist ein mit viel Liebe zum Detail gemachtes System. Der generierte Code ist schnell, die Möglichkeiten der Einflußnahme auf die Generierung sind äußerst vielfältig. Die Entwickler haben sich bei ihrer Arbeit wirklich etwas gedacht. Das zeigt auch die erfreulich umfangreiche Utility Library; der 'normale Tagesbedarf' wird mehr als ausreichend abgedeckt. Man kann sogar 'chainen', wobei auch der Zugriff auf gemeinsame Datenbestände möglich ist.

Die Arbeit mit dem System macht Spaß, trotz des etwas mühsamen Debugging. Das System erfüllt viele der Anforderungen, die an ein Entwicklungssystem gestellt werden. Wer also Lust hat, eine moderne, modular aufgebaute Sprache mit einem modernen Compiler zu erproben: hier liegt er bestimmt nicht falsch. Doch ich möchte es mit dem Autor des Handbuches halten: Man sollte nicht gerade Anfänger sein — oder sich die entsprechende Literatur gleich mitkaufen! Und die Besitzer von Compilern, welche nicht zwischen Groß- und Kleinschreibung unterscheiden, seien vorgewarnt: 'Modula-2 for Z80' tut es! Ich kann Ihnen versprechen: DAS ist ein herrlicher Spaß!

Die Bausteine meiner Tochter betrachte ich jetzt neidlos: meine sind mindestens genauso gut. Es bleibt nur ein echter Wermutstropfen: Warum, in aller Welt, denn nur sieben signifikante Stellen? □

Literatur

[1] M. Bodag, c't Heft 7/84, Der Turbo aus der Westentasche

[2] Byte, Januar 83, S. 283 ff. (Sieb des Eratosthenes)

[3] G. Pomberger, Softwaretechnik und Modula-2, Linz 1984, Hanser Verlag

In Deutschland ist das 'Modula-2 System for Z80 CP/M' bei der Firma Tesco, Rüdenhausener Straße in 8714 Wiesentheid für 499 DM erhältlich.