Der Ada-Trainingskurs auf dem C 64

Ada wurde im Jahre 1975 vom amerikanischen Verteidigungsministerium in Auftrag gegeben, um die Schwierigkeiten und die hohen Kosten in den Griff zu bekommen, welche durch die Vielfalt konkurrierender Programmmiersprachen entstanden sind. Jetzt gibt es auch auf dem C 64 die Möglichkeit, mit dieser modernen Sprache zu arbeiten.

Nur einige wenige Programmiersprachen — sogenannte Hochsprachen — haben eine weite Verbreitung gefunden. Cobol in der Wirtschaft, Fortran und Pascal in der Naturwissenschaft und Basic in der Home-Computerei.

Im riesigen Anwendungsgebiet der Industrie und Technik aber sind Hochsprachen selten zu finden.

Warum? Nun, die meisten für Steuerungen eingesetzten Computer sind nur mit einer bestimmten Hochsprache ausgestattet, die aber spezielle Eigenschaften des Computers oft nicht oder nur umständlich ausnutzt. Eine solche Hochsprache ist daher für diese Zwecke in der Regel nicht genügend effizient, zu langsam und zu aufwendig im Speicherbedarf.

Industrie-Programmierer greifen deshalb viel lieber auf Assemblersprachen zurück, mit dem Ergebnis, daß Programme nicht auf andere Computer übertragbar sind, meistens nur vom Autor selber verstanden und verbessert und nicht zu größeren Programmblöcken zusammengefügt werden können.

Selbst dann, wenn Programmierer die gleiche Hochsprache benutzen, müssen sie oft den Umgang mit neuen Betriebssystemen und anderen Programmentwicklungsmitteln lernen, beispielsweise Text-Editoren, Grafiksysteme, Konfigurationskontrollen und Fehlersuchhilfen. Schließlich sind Programme, die es gestatten, von einem System in ein anderes zu übersetzen, sehr aufwendig und teuer, genauso wie das Um- und Dazulernen der Programmierer.

Kein Wunder also, daß ein so großer Auftraggeber und Selbstverbraucher von Software wie das Verteidigungsministerium in den USA eine Hochsprache forderte, die alle diese Mängel abstellen sollte. Ganz nebenbei sollte diese Sprache stark strukturiert sein, eine einfache Fehlerbehandlung ermöglichen, mit unterschiedlicher Hardware zusammenarbeiten und schließlich auch noch die parallele Bearbeitung von Prozessen erlauben.

Die Antwort auf diese Herausforderung ist Ada: eine neue Standardsprache, die alle anderen Hochsprachen ablösen soll.

Um das zu erreichen, oder besser gesagt, um zu verhindern, daß Ada auch nur wieder eine Hochsprache von vielen wird, wurden die begabtesten Sprachexperten aus Wissenschaft und Industrie zur Entwicklung herangezogen. Diese Entwickler haben der Sprache Ada viele erprobte und bewährte Elemente anderer Hochsprachen einverleibt, mit dem Ziel, »modern software practices«, das heißt also, moderne Methoden der Software-Entwicklung zu ermöglichen.

Das Ada-Konzept

Die prinzipiellen Eigenschaften von Ada lassen sich in mehreren Gruppen zusammenfassen:

1. Definition von Datentypen, wie:
TAGE_IM_MONAT: INTEGER RANGE 20..31;
erzeugt hier eine Variable mit dem Namen »TAGE_IM_MONAT«, welche nur die ganzzahligen Werte von 20 bis 31 annehmen kann. Damit kann der Programmierer seine eigenen Datentypen erfinden und sie dann zur Benennung von Variablen hernehmen.

Die Datentypen können auch ihre eigenen Werte explizit benennen, wie zum Beispiel:
TYPE MONAT IS (JAN,FEB,MAR, APR);
DER_BESTE_MONAT : MONAT;
Dann kann man schreiben:
IF DER_BESTE_MONAT = APR THEN ....

Diese Eigenschaft von Ada erhöht nich nur die Lesbarkeit, sondern erleichtert es dem Programmierer, die Übersicht zu behalten und erlaubt anderen Leuten, das Programm zu verstehen.

2. Das Konzept der Programmstruktur stellt Befehle wie IF..THEN..ELSE..ENDIF und CASE..WHEN..ENDCASE zur Verfügung, welche zusammen mit der Schleifenbildung jeden logischen Programmfluß ohne GOTO zulassen.

3. Um zu erreichen, daß mehrere Leute Programmteile schreiben können, die später zusammenfügbar sind, hat ein Programm getrennte Merkmale »nach außen« (specification) und »nach innen« (body). Zum Beispiel kann ein Programmierer ein Programm zur Quadratwurzel schreiben. Die »specification« definiert genau, was andere Benutzer wissen müssen (welche Angaben es braucht und welche Angaben es liefert). Im »body« können Methoden und Formeln verwendet werden, die niemanden außer den Autoren selbst interessieren.

4. Ada-Programme bestehen aus »Paketen«, das heißt sie sind blockstrukturiert. In einem Paket sind beliebige Daten und Anweisungen zusammengefaßt, die eine ganz bestimmte Aufgabe erfüllen. Ein Paket kann in verschiedenen Systemen verwendet werden. Es ist letztendlich möglich, Software-Pakete genauso in bestehende Systeme »einzustecken«, wie man das heute mit integrierten Chips macht.

5. Schließlich soll Ada auch Multitasking erlauben. Darunter versteht man die Möglichkeit, mehrere Programmteile gleichzeitig und unabhängig voneinander ablaufen zu lasen und sie nur an ganz bestimmten Stellen zusammenzubinden.

Soviel zur Sprache Ada selbst. Was aber ist der Stand der Entwicklung heute?

Ada wird jetzt gerade eingeführt, aber nicht nur vom Militär und nicht nur in den USA.

Europäische Aktivitäten konzentrieren sich auf Frankreich, Deutschland, Finnland und Großbritannien. Es haben sich schon User-Groups gebildet, Universitäten beschäftigen sich mit Anwendungsstudien und viele Organisatio-

nen bemühen sich um die Entwicklung von Lehr- und Trainingsprogrammen.

Eins haben alle diese Aktivitäten gemeinsam: sie beziehen sich ausschließlich auf große und mittlere Computeranlagen und beschränken auf diese Weise den Zugang zu Ada nur auf den exklusiven Kreis der Profis.

Ada auf dem C 64

In dieser Situation bildet der Ada-Trainingskurs für den C 64 von Data Becker eine hochzulobende Ausnahme, denn er führt erstmalig Ada in die Welt der Home-Computer ein. Der Trainingskurs besteht aus einem Handbuch und einer Programmdiskette, mit deren Hilfe Ada so erklärt wird, daß komplette Programme in dieser Sprache geschrieben werden können.

Die Programmdiskette enthält fünf Programme:
— einen Editor
— einen Syntax-Prüfer
— einen Semantik-Prüfer plus Code-Generator
— einen Assembler (Übersetzer)
— einen Disassembler

Für einen Hobby-Programmierer, der nur Basic kennt, ist das sehr verwirrend. Aber ich bitte zu bedenken, daß auch Basic nicht einfach so abläuft, wenn »RUN« eingetippt wird, sondern daß jede Basic-Zeile vom Computer intern in einen komplizierten Vorgang analysiert werden muß. Dieser Analyse-Vorgang ist in den Commodore-Computern fest eingebaut. Für Ada muß der Trainingskurs einen entsprechenden Übersetzer separat zur Verfügung stellen — eben den oben aufgelisteten Code-Generator und den Assembler.

Außerdem ist jedem Basic-Programmierer geläufig, daß der Rechner merkt, wenn ein Befehl falsch geschrieben worden ist, ein Komma fehlt oder in einer Zeile sonst irgend ein Fehler gemacht worden ist. Die nicht immer verständlichen englischen Fehlermeldungen sind ja oft genug Grund zur Frustration.

Diese Überprüfung besorgt beim Ada-Trainingskurs das Syntax- und das Semantik-Prüfprogramm, welche auf Einhaltung der Rechtschreibung und der Grammatik von Ada achten.

Der Editor steht genau wie bei Basic am Anfang aller Aktivitäten. Bei Basic denken wir nicht lange nach und nehmen es als gegeben hin, daß wie per Tastatur Zeichen, Zahlen und Texte schreiben und eingeben können, daß der Cursor bewegt werden kann und daß uns alle möglichen Steuertasten zur Verfügung stehen. Das eingebaute Betriebssystem stellt uns diese Editor-Funktion zur Verfügung.

Für Ada muß das alles extra gemacht werden und dazu dient das Editor-Programm.

Der Disassembler schließlich ist ein Luxus, der zusätzlich zum Assembler angeboten wird, für ein Ada-Programm aber nicht unbedingt benötigt wird. Soviel sei zur Programmdiskette gesagt.

Meine Meinung über das Handbuch ist zweigeteilt. Einerseits finde ich die Einführung in Ada und die Programmierung sehr gelungen, verständlich und gut lesbar. Die Übungsbeispiele sind klar dokumentiert und erläutert.

Andererseits aber sind die Anweisungen, wie die Übersetzungsprogramme zu bedienen sind, vermischt mit einer sehr detaillierten Beschreibung der Arbeitsweise eben dieser Programme, die nur für Fachleute verständlich ist. Das hat zur Folge, daß der Ada-Anwender, der sich nur für die Sprache selbst interessiert, in Erklärungen ertrinkt, welche für das Verständnis und die Anwendung von Ada allein nicht notwendig sind.

Ich selbst habe mehrere Abende gebraucht, um ein erstes kleines Ada-Programm zum Laufen zu bringen und dafür — ich bin so frei — gebe ich dem Handbuch die Schuld.

Zusätzlich enthalten die Teilprogramme einige kleine Unzulänglichkeiten, auf die ich im folgenden noch eingehe.

Das erste Ada-Programm

Zunächst aber möchte ich beschreiben, wie man zum ersten Ada-Erfolgserlebnis kommt.

Zuerst wird der Editor geladen. Er meldet sich nach kurzer Ladezeit von selbst und muß nicht — wie im Handbuch beschrieben — mit »RUN« gestartet werden.

Der Editor, und noch viel mehr die anderen Programmteile, haben zum Teil ziemlich lange Ladezeiten, die weder im Text noch auf dem Bildschirm angekündigt werden. So ist man häufig im Zweifel, ob man noch warten soll, oder ob das Programm abgestürzt ist, was leider zuweilen auch vorkommt.

Erfreulich, weil leicht verständlich und übersichtlich, sind die verschiedenen Menüs.

Abgesehen von anfänglichen Farbeinstellungen zeigt das Editor-Menü die Befehle für zwei Aktionsgebiete an:
— Schreiben beziehungsweise Ändern des Ada-Programmtextes
— Weiterverarbeiten des Ada-Programms, Ein- und Ausgabe

Beide Befehlssätze bedienen sich der Funktionstasten. Da jede Funktionstaste dadurch zwei Bedeutungen hat, muß der Lernende am Anfang ziemlich oft die Menüs aufrufen, um seinem Gedächtnis auf die Sprünge zu helfen.

Der Editor

Das Schreiben und Ändern verfügt über sehr komfortable Hilfsmittel, wie automatische Zeilennumerierung, Einfügen von Zeilen, vor- und rückwärts.

Ein Programm, auch ein unfertiges, kann mit dem zweiten Befehlssatz ausgedruckt oder auf Diskette abgespeichert werden. Ein abgespeichertes Programm ist von Diskette wieder ladbar. Überhaupt stehen alle gängigen Anweisungen an die Diskettenstation zur Verfügung.

Wenn man schließlich glaubt, daß ein Programm fertig und natürlich fehlerfrei ist, wird es zum Übersetzen geschickt.

Die entsprechende Funktionstaste zaubert neue Anweisungen auf den Bildschirm, welche abfragen, ob man wirklich übersetzen will und wenn ja, ob das Programm zu allererst abgespeichert werden soll. Die erste Frage bietet eine belächelbare, die zweite Frage eine sehr sinnvolle Sicherung gegen Programmverlust.

Die dritte Frage nach einer »Spur« erscheint im Textbuch nicht. Ich habe weder sie noch ihre Auswirkungen irgendwo finden können. Natürlich ist es durch Experimentieren letztlich möglich, den Sinn der »Spur«-Entscheidung herauszutüfteln. Aber hier muß die Frage erlaubt sein, ob das im Sinne eines Lernprogramms ist? Für alle Interessierten: Mit der »Spur«-Option wird der Compiler angewiesen, zusätzliche Textangaben, die Auskunft über die gerade abgearbeiteten Zeilen geben, in den erzeugten Code einzucompilieren. Es handelt sich also um eine Art »Trace«-Funktion für Maschinensprache.

Doch weiter: Immer noch als Teil des Editors erfolgt jetzt eine »lexikalische Analyse«, welche (Zitat) »die einzelnen Worte des Programms erkennen und Worte, die in Ada keinen Sinn ergeben, ausfiltert«.

Bevor mitgeteilt wird, ob diese Analyse erfolgreich war oder Feh- ►

ler aufgespürt wurden, wird erst einmal die Programmdiskette verlangt, von welcher der nächste Programmteil, nämlich der Syntax-Prüfer eingelesen und nach einer längeren angsterfüllten Wartezeit gestartet wird.

Die Syntax-Analyse prüft, ob ein Programm den grammatischen Regeln von Ada entspricht.

Ein »lexikalischer« Fehler (»krocedure« statt »procedure«) führt prompt zu einer Fehlermeldung, gefolgt von der Frage, ob der »Stack« ausgegeben werden soll. Im Handbuch ist zwar, eingebettet in eine ausführliche, an den Fachmann gerichtete Beschreibung der Arbeitsweise des Syntax-Prüfers, diese Ausgabemöglichkeit erwähnt aber nicht, wie sie herbeigeführt wird. Mehrfaches Drücken der RETURN-Taste brachte in der Tat Zahlenreihen auf den Schirm, deren Bedeutung zwar im Detail beschrieben wird, dem Laien aber nicht weiterhilft.

Ausgezeichnet dagegen ist die nachfolgende Fehlerdiskussion, welche die fehlerhafte Zeile, den Fehler und mögliche Abhilfen und Korrekturen aufzeigt.

Die Syntax-Prüfung wird abgebrochen, und es kommt die Aufforderung, wieder die Programmdiskette einzulegen. Es leuchtet natürlich ein, daß von ihr wieder der Editor eingelesen wird, um den Fehler beheben zu können. Was nicht einleuchtet, ist, daß nach dem Laden das Programm hängen bleibt: Absturz!

Bei Entdeckung eines syntaktischen Fehlers (zum Beispiel ein fehlendes Semikolon) läuft dieselbe Prozedur ab, jedoch mit zwei großen Ausnahmen. Zum ersten wird nach der Fehlerdiskussion die Syntax-Prüfung fortgesetzt, um noch das restliche Programm zu analysieren. Zum zweiten stürzt das Programm beim Wiedereinladen des Editors jetzt erfreulicherweise nicht ab. Natürlich kann in beiden Fällen das fehlerhafte Ada-Programm wieder geladen werden. Aber ein Absturz ist halt einfach ärgerlich.

Sind alle lexikalischen und syntaktischen Fehler ausgemerzt, folgt die Aufforderung, von der Programmdiskette den dritten Teil, nämlich den Semantik-Prüfer, einzulesen. Dieser prüft, ob das Programm vom Aufbau her korrekt ist. Zum Beispiel ist eine Anweisung an den Drucker, die das fehlerhafte Wort »Dricker« verwendet, lexikalisch und syntaktisch richtig, wird aber von der semantischen Analyse erwischt.

Trotz eventueller Fehler übersetzt der Code-Generator des Semantik-Prüfers in einem ersten Durchgang das Ada-Programm in ein Assemblerprogramm. Dieses kann entweder mit dem Assembler endgültig in ein Maschinenprogramm übersetzt werden, oder es muß mit dem Editor erst noch korrigiert werden. Für Assemblercode-Spezialisten kann auch das Assemblerprogramm geladen und direkt verbessert werden. Nur eins, die Befolgung der Aufforderung, eine dieser drei Optionen zu benützen, führt unweigerlich zum Absturz. Es hat einige Versuche gekostet, bis als einzige Vorgehensweise feststand, den Computer zuerst durch Ausschalten zurückzusetzen — wahrhaft keine elegante Methode in einem Trainingskurs.

Ist das Ada-Programm endlich fehlerfrei, wird der Assembler geladen. Er fragt nach dem Namen des vorläufigen Assemblerprogramms — und wehe, Sie denken nicht daran, unaufgefordert die Datendiskette einzulegen, auf der das besagte, zu übersetzende Programm gespeichert ist. Diese kleine Unaufmerksamkeit wird wieder mit Absturz des Programms bestraft.

Wenn Sie es aber richtig machen, übersetzt der Assembler in einem zweiten Durchgang das Programm in echten Maschinencode, der mit »RUN« gestartet und sonst wie ein normales Maschinenprogramm behandelt werden kann.

Wie eingangs erwähnt, braucht man den Programmteil Disassembler für Ada nicht, es sei denn, der Kursteilnehmer will, was der Autor empfiehlt, gleich die Gelegenheit nutzen und Maschinensprache lernen. Dann allerdings kommt ihm die ausführliche Beschreibung von Assembler und Disassembler, die unabhängig von Ada auf eigenen Füssen stehen, sehr zugute.

Zusammenfassung

Ich bin begeistert von der Möglichkeit, die Sprache Ada zu lernen, und in ihr auf dem C 64 programmieren zu können.

Die Lehrmethode einer schrittweisen Einführung in die Sprache ist auch für Laien geeignet, die bisher nur in Basic gearbeitet haben.

Die Übersetzungsmethode ist zwangsläufig etwas langwierig, die vielen Diskettenwechsel lassen sich durch den begrenzten Speicher des C 64 nicht vermeiden. Und wenn man einmal den Dreh gefunden hat, ist das nicht mehr störend.

Bis dahin allerdings ist es ein steiniger Weg. Die Programme sind leider nicht laiensicher. Ein Lernprogramm muß einfach dem dümmsten Bedienungsfehler Rechnung tragen und nicht, wie in diesem Trainingskurs, immer wieder zum Absturz führen.

Der begreifliche Stolz des Autors auf die für meine Begriffe fantastische Leistung, die Prüf- und Übersetzungsprogramme geschrieben zu haben, drängt die Beschreibung derselben zu sehr in den Vordergrund. Auch das Argument, daß dadurch wichtige Grundkenntnisse der Assembler- und Maschinensprache erlernbar sind, zieht meines Erachtens nicht. Denn mit dem Trainingskurs — für den stolzen Preis von 198 Mark — will ich Ada lernen und nicht Assemblersprache.

Es wäre Data Becker dringend zu empfehlen, sowohl das Handbuch als auch die Programme zu revidieren und davon eine zweite Version herauszugeben. Die kritisierten Mängel sind alle leicht korrigierbar, wenn man sich die Zeit nimmt und die Bedürfnisse von Laien vor Augen hält.

Dieses ausgezeichnete Unterfangen, die Zukunftssprache Ada den Home-Computern zu eröffnen, darf einfach nicht an ärgerlichen Programmiermängeln und inkonsequenten Erklärungen scheitern.

Ada — für wen?

Der Ada-Trainingskurs kann allen an modernen Programmiersprachen interessierten Anwendern empfohlen werden. Auch wer sich für Compilerbau interessiert, ist mit diesem Kurs gut beraten, da das Handbuch sehr ausführlich auf allgemeine Prinzipien der Compilierung eingeht. Allerdings sollte man die Fähigkeiten dieser Ada-Trainingsversion nicht überschätzen. Der Ada-Trainingskurs eignet sich entschieden besser zum Lernen von Ada als für größere Programmierungsprojekte.

Nicht verschwiegen werden soll auch, daß eine Ada-Version für einen Heimcomputer immer nur die wichtigsten Aspekte der Sprache unterstützen kann. So unterstützt Ada als Trainingskurs beispielsweise kein Multitasking, also keine Parallelverarbeitung bestimmter Programmabschnitte. Doch damit kann man leben. Entscheidend ist die Möglichkeit, eine moderne Sprache wie Ada mit dem C 64 lernen zu können. (Dr. Helmuth Hauck/ev)