wdv-notes 235
Stand: 2.FEB.1994 (2.)
Wiss.Datenverarbeitung
FREIE UNIVERSITÄT BERLIN

VAX C: Erste Hilfe für C-Portierungen.

© 1993–1994 Edited by *Karl-Heinz Dittberner*

Pro

Die Programmierung in der Sprache »C« ist bekanntlich mit einigen Problemen behaftet [1]. *Betroffen davon ist auch die Programmierung in* ***VAX C****, einem C-Compiler, der im wesentlichen der (ursprünglichen) Sprachdefinition von Kernighan & Ritchie sowie ANSI-Empfehlungen folgt* [6, 7]. *Sollen nun unter* ***VAX C*** *entwickelte Programme zu anderen Plattformen portiert werden, so sollten rechtzeitig einige spezielle Eigenschaften und Besonderheiten des* ***VAX C*** *beachtet werden.*

In einem Beitrag gibt Thomas Horn in der Fachzeitschrift »Deckblatt« im Oktober 1993 einige sehr gute Hilfestellungen zum Arbeiten mit dem ***VAX C****. Dieser Artikel wird im folgenden wegen der grundsätzlichen und kostensparenden Bedeutung – insbesondere zur Fehlervermeidung bei Portierungen – vollständig wiedergegeben.*

So vermeiden Sie Stolpersteine

Thomas Horn über die Anwendung des VAX C Compilers

Aus: »Deckblatt« – Oktober 1993

Bei der Arbeit mit dem VAX C Compiler gilt es, bereits im Vorfeld auf ein paar Feinheiten aufzupassen, damit sich eine spätere Portierung der entwickelten Software einfach realisieren läßt. Kennt man die Stolpersteine und meidet sie, sind beispielsweise unter *vcc* entwickelte Programme relativ problemlos nach OpenVMS zu bringen. Wie dies zu bewerkstelligen ist, erklärt dieser Beitrag.

Der *VAX C* Compiler folgt im wesentlichen der Sprachdefinition von Kernighan & Ritchie. Außerdem enthält der *VAX C* Compiler eine Reihe von Erweiterungen, die einem frühen Entwurf der ANSI C-Norm (bzw. dem XPG3-Standardentwurf der X/Open-Gruppe) entnommen wurden, sowie weitere zusätzliche VMS-spezifische Erweiterungen.

Da sehr viel Software unter *VAX C* für VMS entwickelt wurde, stellt Digital auch einen *VAX C* Compiler (*vcc*) für Ultrix, speziell auch für Ultrix/RISC, zur Verfügung [8]. Der *vcc* enthält allerdings auch einige kleinere Einschränkungen, bei deren Beachtung jedoch eine problemlose Übernahme von *VAX C* Programmen unter Ultrix möglich ist. Einige Erweiterungen der Sprache »C« unter *vcc* sind im allgemeinen problemlos, können aber Schwierigkeiten hervorrufen, wenn Software unter dem *vcc* entwickelt wird, die dann auch zum OpenVMS portiert werden soll.

Deshalb sollte für die Neuentwicklung von Software besser der *DEC C* Compiler benutzt werden, der ein „sauberes" ANSI-C unter beiden Betriebssystemen realisiert. Falls jedoch aus verständlichen Kompatibilitätsgründen weiterhin der *VAX C* Compiler genutzt werden muß, sollte man auf einige spezielle Eigenschaften, Besonderheiten und Erweiterungen des *VAX C* achten, um spätere Portierungen nicht unnötig zu erschweren.

Aufgrund der 32-Bit Architektur der VAX und der damit verbundenen virtuellen 32-Bit Adressen entsprechen die Schlüsselwörter *int* und *unsigned* dem 32-Bit Format und damit den Schlüsselwörtern *long int* bzw. *unsigned long*. Das 16-Bit Format wird nur von *short* bzw. *unsigned short* realisiert. Zur besseren Gewährleistung der Kompatibilität der *VAX C* Programme zu anderen »C«-Systemen (z. B. Microsoft C, Borland-C, PDP-11 C, usw.) sollte weitgehend auf die Anwendung von *short* oder *long* bzw. *unsigned short* oder *unsigned long* orientiert werden.

Nur wo eine prozessorabhängige Auswahl der Datenformate gerechtfertigt ist, sollte *int* oder *unsigned* verwendet werden. Damit haben die Schlüsselwörter *int* und *unsigned* ihre frühere Bedeutung weitgehend verloren. Diese Bemerkung trifft gleichermaßen auch auf die »C«-Systeme unter den Ultrix/Unix-Systemen zu.

Für die Deklaration von Gleitkommazahlen gibt es in »C« die Schlüsselwörter *float* und *double*, wobei *float*-Datenobjekte normalerweise intern im F-Format (4 Byte) und *double*-Datenobjekte im D-Format (8 Byte) dargestellt werden.

Vorsicht bei den C-Keywords »int« und »unsigned«!

Zur Darstellung von Zahlen mit einer größeren Genauigkeit ist im D-Format die Mantisse um vier Byte erweitert. Durch die hardwaremäßige Stützung der Gleitkommaoperationen über den FPA (Floating Point Accelerator) ergeben sich für das kürzere F-Format keine Zeitvorteile. Deshalb werden im *VAX C* Compiler standardmäßig durch den Qualifizierer /PRECISION = DOUBLE die *float*-Datenobjekte den *double*-Datenobjekten gleichgesetzt [2].

Auf das G-Format (8 Byte) der VAX kann zurückgegriffen werden, wenn beim Compilieren der Qualifizierer /G_FLOAT angegeben wird, wobei sowohl *float*- als auch *double*-Datenobjekte im G-Format dargestellt werden. Bei Benutzung des G-Formats ist beim Linken die Laufzeitbibliothek für das G-Format VAXCRTLG.OLB zu verwenden. Das H-Format wird vom *VAX C* Compiler nicht unterstützt.

Bitfelder sind im *VAX C* auf 32 Bit begrenzt. Die Bits werden von rechts nach links angeordnet. Zur optimalen Verbindung von C-Programmen und Assembler-Programmen gibt es im *VAX C* zusätzliche Speicherklassenbezeichner:

- *globaldef* — Definition eines globalen Datenobjekts,
- *globalref* — Deklaration einer globalen Bezugnahme,
- *globalvalue* — Definition einer globalen Konstanten.

Mit *globaldef* werden die globalen Datenobjekte vereinbart und in einem besonderen Programmabschnitt abgelegt. In geschweiften Klammern kann der Programmabschnittsname als Zeichenkette spezifiziert werden, z. B.:

```
globaldef {"common1"} double
tabelle[MAX];
```

Zusätzlich können die Speicherklassenmodifikatoren *noshare* und *readonly*

C-Speicherklassenbezeichner »globaldef, globalref, globalvalue« müssen richtig verwenden werden.

angegeben werden. Der Modifikator *noshare* bewirkt, daß bei gemeinsam nutzbaren Programm-Images ein bezugnehmendes Programm von dem Programmabschnitt aus dem gemeinsam nutzbaren Programm-Image seine eigene Kopie erhält. (In der ANSI-C Norm entspricht *readonly* dem Schlüsselwort *const*).

Der Modifikator *readonly* verbietet den Schreibzugriff zu dem Programmabschnitt, z. B.:

```
globaldef {"common2"} readon-
ly int tab[]={1,2,5,8};
```

Die Bezugnahme auf globaldefinierte Datenobjekte aus anderen Programm-Modulen erfolgt über den Speicherklassenbezeichner *gobalref*, z. B.:

```
gobalref double tabelle[];
```

Die Querverbindung zwischen den Modulen wird vom Linker hergestellt.

Der Speicherklassenbezeichner *globalvalue* kann verwendet werden, um Integer-Symbole einen Wert zuzuweisen. Diese Integer-Symbole belegen keinen Speicherplatz zur Laufzeit, da sie nur vom Linker benutzt werden, um in anderen Programmen Werte zu substituieren, z. B.:

```
globalvalue code=100;
main()
{ …
if(a==code) …
  …  }
globalvaluecode;
test()
{ …
if (x!=code) …
  … }
```

Der Linker substituiert im gesamten Programmsystem *code*() durch 100. Die zu startende Funktion muß gemäß Sprachdefinition den Namen *main* tragen (*Anm. des Editors:* So „beschränkt" ist eben das so beliebte »C«). Der Funktionsname ist aber gleichzeitig der globale Eintrittspunktname, was zu Fehlern führt, wenn man mehrere Hauptprogramme in eine Objektmodul-Bibliothek einfügen will, da jedes globale Symbol nur einmal auftreten darf. Deshalb ist im *VAX C* eine zusätzliche Option *main_program* implementiert, die anstelle von *main* beliebige Hauptprogrammnamen zuläßt, z. B.:

```
test_1(argn,argv) main_program
int argn; char *argv[];
```

„Mit C ist es weitaus schlimmer: Man kann einfach alles damit machen, wenn man gerissen ist." – N. Wirth 1991

Eine weitere Ergänzung ist der Zugriff auf Textbibliotheken durch die *#include*-Anweisung. Standardmäßig sind in »C« realisiert:

```
#include "file"
#include <file>
```

Im ersten Fall wird die Datei im Verzeichnis gesucht, das das Quellenprogramm enthält. Ist die Suche ergebnislos, so wird der logische Name C$INCLUDE, falls vorhanden, für die Definition des Verzeichnisses benutzt. Im zweiten Fall wird das Verzeichnis vom logischen Namen VAXC$INCLUDE übernommen. Wenn der logische Name nicht definiert ist, so wird die Datei im Standardverzeichnis SYS$LIBRARY gesucht. Die Erweiterung der *#include*-Anweisung lautet:

```
#include modul
```

Das Modul *modul* wird in den Textbibliotheken gesucht, die im *cc*-Befehl (Start des *VAX C* unter OpenVMS) spezifiziert wurden [2]. Wenn das Modul nicht gefunden wurde, so wird die Textbibliothek durchsucht, die dem logischen Namen C$LIBRARY, falls vorhanden, zugeordnet ist. Wenn auch diese Textbibliothek das Modul nicht enthält, so wird in der Standardbibliothek SYS$LIBRARY:VAXCDEF.TLB gesucht.

Da die Arbeit mit Bibliotheken effektiver ist, werden die Definitions-Files (wie stdio.h usw. [3]) in der Regel in der Standardbibliothek und nicht als separate Quellendatei zur Verfügung gestellt. Deshalb müssen meistens die *#include*-Anweisungen in den Quellenprogrammen, die von anderen »C«-Systemen übernommen werden, modifiziert werden, indem man die spitzen Klammern in vorhandenen Quellprogrammen entfernt, z. B.:

```
#include stdio
```

Andererseits ist das aber in den meisten Fällen die einzige erforderliche Nacharbeit, wenn ein Programm vom *VAX C* portiert werden soll. Aus Portabilitätsgründen sollte heute bei einer Programmentwicklung unter *VAX C* auf die Bibliotheksanwendung im Regelfall verzichtet werden. Dafür spricht auch, das *vcc* unter Ultrix diese VMS-spezifischen Bibliotheken nicht unterstützt.

Weiterhin sei noch auf eine häufige Fehlerquelle hingewiesen. Im *VAX C* sind aus Kompatibilitätsgründen auch die alten Zuweisungsoperationen (=+, =–, =*, =/, =& usw.) implementiert. Deshalb sollte man sich zur Vermeidung von Fehlern prinzipiell angewöhnen, nach dem Zuweisungsoperator ein Leerzeichen zu schreiben, z. B.:

a=–2; entpricht *a*–=2;
und nicht a= –2;

Wesentlich kritischer ist noch:

p=&*a*; entpricht *p*&=*a*;
und nicht *p*= &*a*;

Wenn *p* korrekt als Pointer (Zeiger) vereinbart ist, bringt der Compiler bei der vermeintlichen Adreßzuweisung, die als Konjunktion interpretiert wird, einen hartnäckigen Fehler (*Anm. d. Editors:* Das C-Chaos ist perfekt, denn der C-Compiler sagt dazu keinen Mucks [1]).

Ein weiterer Hinweis betrifft die Bildung von Identifikatoren (Bezeichnern) im *VAX C*. Gegenüber der Sprachdefinition von Kerninghan & Ritchie [4] wurde im *VAX C* die signifikante Länge von Bezeichnern auf 31 Zeichen (wie im ANSI-C) erweitert. Der *vcc* unter Ultrix unterstützt dagegen schon Bezeichner mit einer maximalen Länge von 255 Zeichen (auch für externe Bezeichner).

„Privat kann jeder soviel C programmieren oder Videos ansehen, wie er mag." – N. Wirth 1991

Als zusätzliche Alphazeichen ist neben dem Underline (_) auch das Dollarzeichen ($) zulässig. Es ist für die Bildung der Systemsymbole des VMS reserviert. Generell gilt gemäß der Norm, daß Klein- und Großbuchstaben eine unterschiedliche Bedeutung haben. Jedoch bei allen globalen bzw. externen Symbolen wird diese Unterscheidung nicht gemacht, da die Dienstprogramme wie Linker und Librarian nicht zwischen Klein- und Großbuchstaben unterscheiden können.

Im *VAX C* sind darüberhinaus folgende ANSI-C Eigenschaften implementiert:

1) Funktionsprototypen gemäß dem ANSI-C Stil, z. B.:

   ```
   float test(char *string,
   short n)
   ```

   an Stelle des Kernighan/Ritchie-Stils:

   ```
   float test(string, n)
   char *string; short n;
   ```

2) Der typenlose Zeigertyp *void** an Stelle der früheren Vereinbarung *char**.
3) Datentyp-Attribute *const* und *volatile*. *const* entspricht dem *VAX C*-spezifischen Datentyp-Attribut *readonly*, das nicht mehr benutzt werden sollte.
4) Hexadezimale Zeichenkonstanten *\xnnn*.
5) Die Reservierung leerer Etiketten für Strukturen und Vereinigungen, z. B.:

   ```
   struct name;
   ```

   wobei die Struktur *name* später definiert werden kann. Der Namensraum der Etiketten von Strukturen und Vereinigungen ist getrennt vom Namensraum der Bezeichner.
6) Die Makros *__DATE__*, *__TIME__*, *__FILE__* und *__LINE__* des Makro-Präprozessors.
7) Die *#pragma*-Anweisung für die Festlegung integraler Wortgrenzen für Strukturen und Komponenten von Strukturen:

   ```
   #pragma [no]member_alignment
   ```

Das Speicherklassen-Attribut *noshare* für das Programmabschnittsattribut NOSHR und die Makro-Präprozessoranweisung *#module* für die Festlegung von Objektmodul-Namen sind kein ANSI-C Standard und werden vom *vcc* beispielsweise nicht mehr unterstützt.

Darüberhinaus unterscheiden sich der *VAX C* Compiler unter OpenVMS und der *vcc*-Compiler unter Ultrix zum Teil sehr wesentlich in den Qualifizierern bzw. Optionen. Vor ihrer Anwendung sollte man sich deshalb darüber in den einschlägigen Dokumentationen ausführlich informieren.

Weitere Probleme können bei einer Portierung von *VAX C* Programmen nach Ultrix (oder Unix) dadurch entstehen, daß unter Ultrix die entsprechenden Ultrix-spezifischen Headerfiles benutzt werden, die manchmal gegenüber dem VMS abweichende Vereinbarungen enthalten, die nirgends (!) dokumentiert sind. Hier hilft im Einzelfall nur ein Vergleich der C-Texte der betroffenen Headerfiles.

Im allgemeinen läßt sich aber feststellen, daß der Portierungsaufwand für *VAX C* Programme von OpenVMS nach Unix bei einer relativ „sauberen" Programmierung ziemlich gering ist.

(Dr. Thomas Horn/te)

Literatur

[1] Dittberner, K.-H.: Der Problem-Fall »C«. FU Berlin (IfP): wdv-notes Nr. 220, 1991–1993.

[2] Dittberner, K.-H.: Das Arbeiten mit C auf dem HAIRcluster. FU Berlin (IfP): wdv-notes Nr. 28, 1989–1992.

[3] Dittberner, K.-H.: VAX C: Liste der verfügbaren Definitions-Files. FU Berlin (IfP): wdv-notes Nr. 193, 1991–1993.

[4] Kernighan, B. W. und Ritchie, D. M.: Programmieren in C. München: Carl Hanser 1983.

[5] Kernighan, B. W. und Ritchie, D. M.: Programmieren in C mit ANSI C. München: Carl Hanser & Prentice Hall 1990.

[6] DEC: Guide to VAX C. Maynard: Digital Equipment Corp. 1987. – Best.-Nr.: AI-L370C-TE. – [CHB 563/1].

[7] DEC: VAX C Run-Time Library Reference. Maynard: Digital Equipment Corp. 1987. – Best.-Nr.: AI-JP84A-TE. – [CHB 564/1].

[8] DEC: Guide to VAX C for Ultrix. Maynard: Digital Equipment Corp. 1988. – Best.-Nr.: AA-ME83A-TE.

Anmerkung des Editors: *Die hier gegebenen Hinweise sind auch für ganz andere Portierungen von C-Programmen – z. B. nach POSIX oder Win32 (Windows NT) – eine sehr wertvolle Hilfe.*

© khd-research • 2004 • Eine Publikation von http://www.khd-research.net/ • File: Notes/wdv-notes_235.pdf