C — die Sprache der Profis

Wenn man Programmierer fragt, welche Computersprache in Zukunft die größte Bedeutung haben wird, beschränken sich immer mehr von ihnen darauf, einen Buchstaben zu nennen: C.

Immer mehr Software-Häuser gehen dazu über, Software in C zu programmieren, ja sogar komplexe Betriebssysteme wie Unix und Gem wurden in dieser Sprache entwickelt. Was ist nun der Grund dafür, daß Programmierer sich für C entscheiden und Pascal, Forth, Cobol oder Maschinensprache links liegen lassen?

Häufige Meinungen der Verfechter von C sind:

Basic ist für professionelle Programme untragbar, da es einfach zu langsam und nicht genügend leistungsfähig ist.

Pascal besitzt eine viel zu strenge Syntax und ist — einmal abgesehen von Turbo-Pascal — eher ein theoretisches Lehrobjekt als eine verwertbare Programmiersprache

Assemblerprogramme sind auf Computer mit anderen Mikroprozessoren kaum übertragbar und schwierig zu schreiben.

Lisp und Prolog sind die »Gralssprachen« der Künstlichen Intelligenz (KI) und für »normale« Programme nur beschränkt verwendbar.

C ist schnell, flexibel, universell

C besitzt keinen dieser Nachteile, bietet aber dem Programmierer eine ganze Reihe von Vorzügen: C ist gleichzeitig eine Hochsprache und eine niedrige, maschinennahe Sprache! Paradox? Nicht unbedingt, C beweist ja, daß das möglich ist. So gibt es umfangreiche Kontrollstrukturen wie IF-ELSE, WHILE, DO-WHILE und SWITCH-CASE und komplexe Datentypen ähnlich den Records in Pascal; gleichzeitig kann man aber auch Systemprogramme schreiben, die direkt auf die Hard- und Firmware des Computers zurückgreifen — und zwar in einem Umfang, wie es sonst nur in Maschinensprache möglich ist. Ein weiterer entscheidender Vorteil ist die Schnelligkeit der erzeugten Programme. Die C-Compiler produzieren entweder mnemonischen Quellcode für Maschinenprogramme oder gleich fertigen Maschinencode. Dieser Code ist sehr schnell und fast so kompakt, als wenn das Programm gleich in Maschinensprache geschrieben worden wäre.

Für Softwarehäuser wohl ausschlaggebend ist die praktisch vollständige Portabilität der C-Programme. So kann ein Programmierer fast ohne Schwierigkeiten C-Software, die auf einem Commodore entwickelt wurde, auf einen IBM-PC, Atari 520-ST oder gar auf Computer der 50000 Mark-Klasse übertragen.

C wurde aus CPL und BCPL entwickelt. Zuerst gab es CPL, die Combined Programming Language, ein Sprachenmonster, das so umfangreich war, daß sich die Programmierer darin nicht mehr auskannten. Der Programmierer Martin Richards von der Universität Cambridge entschloß sich aus diesem Grund, alles irgendwie Entbehrliche von CPL wegzulassen und schuf damit BCPL, die Basic Combined Programming Language. Vielen war auch BCPL noch zu umfangreich, und so entwickelte Ken Thompson von den US-amerikanischen Bell-Laboratories »B«, den direkten Vorfahren von C. B war eine äußerst knappe Sprache, die sich aber sehr gut zur Systemprogrammierung, dem geplanten Einsatzgebiet, eignete. Doch B war schon wieder zu spezialisiert. Und so erinnerte sich Thompsons Kollege Smith an BCPL und entwickelte C.

Das erste C-Programm

Schauen wir uns ein einfaches C-Programm an:

```
main ()
 {
  printf("So sieht ein C-Programm
aus!");
 }
```

»Main« ist der Name der Hauptfunktion, der einzigen benutzerdefinierten Funktion in diesem Miniprogramm. In Klammern kann nach dem Namen ein Parameter übergeben werden, mit dem die Funktion rechnen kann. Vergleichbar in Basic wäre der Befehl PRINT SIN(3). Hier fungiert die 3 als Parameter, damit diese trigonometrische Funktion weiß, wovon sie den Sinus berechnen soll. Da das C-Programm keinen Parameter benötigt, folgen dem Funktionsnamen leere Klammern.

»printf« ist schon die erste Funktion, die Sie verwenden, obwohl Sie gar nicht wissen, wie sie funktioniert: C kennt keinerlei Ein- oder Ausgabebefehle, die WRITE und READ in Pascal oder INPUT und PRINT in Basic vergleichbar wären! Solche Funktionen müssen dem Compiler durch Bibliotheken zur Verfügung gestellt werden. Diese Bibliotheken sind Dateien, die schon beim Kauf eines Compilers mitgeliefert werden. Meistens sind diese wichtigen Funktionen unter dem Namen STDIO auf der Programmdiskette zu finden. STDIO steht für »Standard Input/Output«. Mit #include STDIO.H und #include STDIO.LIB können Sie dem Compiler mitteilen, daß er diese Dateien in den Programmcode einbinden soll.

»printf« erlaubt die formatierte Ausgabe von Daten, hier einer Stringkonstanten. Die C-Programme selbst werden von geschweiften Klammern umgeben. Innerhalb der Klammern stehen alle Variablendefinitionen und Programmbefehle der Funktion.

Listing 1a zeigt, auf welche Weise Variablen definiert werden können. Hier werden zuerst zwei Integervariable als »zahl« und »zahly« bezeichnet, eine Zeichenvariable wird »buchstabe« genannt. Den beiden numerischen Variablen wird gleichzeitig der Wert 5 zugewiesen. In Basic würde diese Zeile etwas anderes bedeuten: Der Computer prüft, ob »zahly« den Wert 5 hat. Trifft dies zu, so wird »zahlx« auf logisch Eins (—1) gesetzt, ansonsten auf logisch Null (0). Also aufpassen, solche Stolperfallen gibt es immer wieder!

Wenn Sie das Programm compilieren, meldet der C-Compiler keinen Fehler; starten Sie aber den Objektcode, dürften Sie ziemlich überrascht sein: Statt zweier Zahlen und des Buchstaben »T« erscheint eine Reihe sinnloser Grafikzeichen! Das liegt daran, daß bei einem formatierten Ausdruck (printf heißt »print formated«, »drucke formatiert«) eine Stringkonstante zur Beschreibung

des Formats angegeben werden muß, wie zum Beispiel bei PRINT USING in Basic eine Reihe von Doppelkreuzen. Ändern wir also die Zeile ab:

```
printf("%d %d\t%c\n",zahlx,
zahly,buchstabe);
```

Wenn Sie jetzt das Programm neu übersetzen, erhalten Sie die erwünschte Ausgabe:

```
5  5   T
```

Was aber bedeuten nun die komischen Prozentzeichen und umgekehrten Divisionsstriche in unserem Print-Befehl? Sie bestimmen das Ausgabeformat (Tabelle 1).

Steuerbefehle

Kaum ein Programm wird von Anfang bis Ende der Reihe nach abgearbeitet; vielmehr ist es immer wieder nötig, bestimmte Werte zu prüfen und ausgehend vom Resultat Entscheidungen zu fällen. C bietet eine ganze Reihe solcher Steuerbefehle, allen voran das aus vielen Sprachen wohlbekannte If-Then-Else. In C kann man das folgendermaßen formulieren:

```
main ( )
 {
  int a;
  a=3;
  if (a = = 3) printf("A hat den Wert
  3\n");
  }
```

Zum Vergleichen zweier Variablen gibt es alle Operatoren, die auch aus Basic bekannt sind. Sie sehen nur etwas anders aus: = = bedeutet gleich, < kleiner, > größer, < = kleiner oder gleich, > = größer oder gleich, ! = ungleich.

Nach dem IF kann immer nur ein Befehl ausgeführt werden. Hier ist es die Funktion »printf«. Mehrere Befehle müssen mit geschweiften Klammern zu einer Verbundanweisung zusammengefaßt werden:

```
if (a = = 3) { printf ("A ist 3!\n";
printf("Und A ist nicht 5!")}
```

Auch ein Befehl, der ausgeführt wird, wenn die Bedingung nicht zutrifft, kann angegeben werden:

```
if (a == 3) printf("A ist 3!")
else printf("A ist nicht 3!");
```

Komplizierte IF-ELSE-Konstruktionen lassen sich oft durch SWITCH UND CASE ersetzen (Listing 1b).

Der SWITCH-Befehl sagt dem Computer, daß die angegebene Variable (hier »var«) untersucht werden soll; CASE prüft, ob ein bestimmter Wert zutrifft und führt in diesem Fall den angegebenen Befehl aus. BREAK verläßt die SWITCH/CASE-Anweisung und ist nötig, damit nicht auch noch die übrigen Möglichkeiten durchgeprüft werden, wenn schon eine Übereinstimmung gefunden wurde. Trifft keine der Bedingungen zu, führt das Programm den unter DEFAULT stehenden Befehl aus (etwa mit dem ELSE bei IF-ELSE zu vergleichen). Die Angabe einer DEFAULT-Bedingung ist optional; wenn der Computer keine findet, fährt er mit der Programmabarbeitung normal fort.

a)

```
main ( )
 {
  int zahlx,zahly;
  char buchstabe;
  zahlx=zahly=5;
  buchstabe='T';
  printf(zahlx,zahly,buchstabe);
 }
```

b)

```
main( )
 {
  int var;
  var=3; /* oder 4 oder ein anderer Wert */
  switch(var)
   {
   case 3: { printf(»VAR ist 3!«); break; }
   case 4: { printf(»VAR ist 4!«); break; }
   default: printf(»Weder 3 noch 4!«);
   }
 }
```

c)

```
main( )
 {
   int loop;
   loop=32;
   while (loop<255)
   {
    printf("%d = %c\n",loop,loop);
    loop=loop+1;
   }
 }
```

d)

```
main( )
 {
  int loop;
  loop=32;
   do
    {
    printf("%d = %c\n",loop,loop);
    loop=loop+1;
    }
  while (loop<255);
 }
```

Listings 1a bis 1d. Verschiedene Beispielprogramme in C

%d	»Decimal«, Dezimalzahl
%x	»heXadecimal«, Hexzahl
%o	»Octal«, Oktalzahl zur Basis 8
%f	»Float«, Fließkommazahl
%e	»Exponential«, Fließkommazahl in Potenzschreibweise
%c	»Character«, Ausgabe als Buchstabe
%s	»String«, Ausgabe des Strings, auf den die Variable zeigt

Tabelle 1. Bedeutung der »Prozent-Variablen« in C

FOR- und WHILE-Schleifen

Auch Schleifen lassen sich auf mehrere Arten programmieren. Zuerst gibt es einmal die FOR-Schleife, die drei Angaben benötigt:
FOR (Anfangswert; Abbruchbedingung; Wertveränderung)
So läßt sich zum Beispiel der ASCII-Zeichensatz ausgeben

```
main ()
{
int loop;
for (loop=32; loop<255; loop=loop+1)
printf("%d = %c\n",loop,loop);
}
```

Die WHILE-Konstruktion benötigt nur ein Argument und wird so formuliert:
WHILE (Bedingung)

Auch mit einem WHILE-Konstrukt kann man den Zeichensatz darstellen. Wie das geht, zeigt Listing 1c.

Pascal-Programmierer wissen, daß es in dieser Sprache neben WHILE auch noch REPEAT-UNTIL gibt. Selbstverständlich kann C das auch. Der Unterschied beider Schleifenkonstrukte liegt darin, ob die Abbruchbedingung vor oder nach der Ausführung der Befehle in der Schleife geprüft wird: WHILE testet die Bedingung vor der Schleife, DO-WHILE erst danach (Listing 1d). Die DO-WHILE-Schleife wird mindestens einmal durchlaufen, auch wenn die Bedingung schon vor dem Eintritt in die Schleife nicht zutrifft.

Ein Small-C-Entwicklungssystem — Editor, Assembler, Linker, Tools zur Textverarbeitung — wird mit C-Quellcode für den C 128 und 128 D von Markt & Technik angeboten.

C auf dem C 64

Der C-Compiler-64 von Data Becker bietet die Möglichkeit, auch auf dem C 64 mit dieser Programmiersprache zu arbeiten. Der Compiler erkennt den Kern der C-Sprache. Die mitgelieferte Funktionenbibliothek ist nicht sehr umfangreich. Der Programmierer muß sich die Funktionen, die er benötigt, zum größten Teil selbst schreiben. Wenn man über ein einziges Laufwerk verfügt, ist die Bedienung des C-Compilers umständlich. Das im Editor erstellte C-Programm wird auf die Original-Diskette gespeichert und dort übersetzt. Um Platz zu schaffen, muß man daher Sourcefiles löschen und später wieder kopieren. Maschinennahe Befehle können ebenfalls nicht ausgenutzt werden. (Martin Kortulla/cg)

Info: C-Compiler-64, Data Becker, Merowinger Str. 31, 4000 Düsseldorf 1, Tel. (02 11) 31 00 10, 298 Mark
Small-C-Entwicklungssystem für C 128 und C 128 D, Markt & Technik Software Vertrieb, Hans-Pinsel-Str. 2, 8013 Haar bei München, Tel. (089) 46 13-220, 148 Mark