Vorwort

Motivation für die Entwicklung der Sprache C war die Existenz des UNIX® Betriebssystems, das ab 1970[1] bekannt wurde. Es sollte erreicht werden, daß UNIX auf andere Prozessoren portiert werden konnte, ohne jedesmal das Betriebssystem vollständig in Assembler neu schreiben zu müssen, sondern eher nur zu 5 %. Die Kommandos dieses Betriebssystems konnten dann gänzlich ohne Neuprogrammierung erzeugt werden – einfach durch Kompilierung.

Deshalb ist bis heute traditionell ein C-Entwicklungssystem für jedes UNIX-System vorhanden. Oft ist es ohne C gar nicht erhältlich.

Lehrgrundlage für dieses Buch waren die beiden Bücher *Programmieren in C* [5] der Autoren Kernighan/Ritchie, wobei Dennis M. Ritchie eher der Erfinder der Sprache C und Brian W. Kernighan eher der Buchautor ist.

Diese Bücher gelten als *C-Bibel* und werden in der C-Szene mit K&R1 und K&R2 bezeichnet. Die Sprache C im Sinne dieser Bücher wird K&R-C bzw. ANSI-C genannt. Ein K&R3 aufgrund des neuen C-Standards von 1999 ist leider nicht in Sicht. Vielleicht ist *dieses* Buch ein gewisser Ersatz dafür.

Die Sprache C war 1974 fertig entwickelt. Das englische Original des Buches K&R1 erschien dann 1978. Ab 1983 gab es Standardisierungsbemühungen, die zum ANSI-Standard C89 und dann auch zum internationalen Standard ISO/IEC C90 führten. Am 1.Dez.1999 wurde der nun aktuelle C-Standard C99 veröffentlicht, der bis dahin mit C9X bezeichnet wurde.

Der aktuelle Standard [4] wurde als Informationsquelle bei der Erstellung dieses Buches benutzt. Beispielkode wurde einem früheren Entwurf des Standards [3] entnommen.

[1] Ein Zeitstempelwert von 0 s unter UNIX ergibt: 01.01.1970 00:00:00

Der C-Standard ist ein englischsprachiges Dokument mit etwa 560 Seiten, das die Sprache C in einer Art *juristischen Vertragssprache* beschreibt, sich eher an Entwickler von Compilern wendet und somit als Lehrbuch oder Tutorial/Kompendium ungeeignet ist. Es wird daher auch nicht daraus zitiert.

Bei Befragungen von Dennis Ritchie[2] sagte dieser beispielsweise auf die Frage, wie ihm der neue C-Standard C99 gefällt, daß ihm der alte Standard C89 gefiel. Er hätte sich gewünscht, daß das Komitee mehr Standhaftigkeit gezeigt hätte gegenüber den vielen Vorschlägen neuer C99 Merkmale. Er kritisiert den beträchtlich gesteigerten Umfang des neuen Standards, und beispielsweise den neuen Typ `_Complex`, der in Folge die Libraries sehr aufbläht.

Es existiert Ende 2004, fünf Jahre nach Veröffentlichung von C99, auch kein einziger C-Compiler, der C99 zu 100 % unterstützt. Bei dieser Bewertung muß bedacht werden, daß ja alle Bibliotheken und alle Header-Dateien gemäß C99 dabei sein müssen. Aber zum Glück unterstützen einige Compiler C99 annähernd zu 100 %. Darunter auch der sehr bekannte `gcc`. C99 ist zwar kein Flop, aber die Stimmung darüber ist bedeckt.

Dieses Buch wendet sich an Leser, die bereits programmieren können und C oder eine andere Programmiersprache ausreichend (oder besser) beherrschen. Unter dieser Voraussetzung ist es sicher möglich, C anhand dieses Buches zu erlernen oder seine Kenntnisse (stark) zu verbessern.

Es liegt die Erfahrung vor, daß wenn Programmierer mit einiger Vorerfahrung eine weitere, für sie neue Programmiersprache erlernen wollen, sie diese neue Sprache möglichst schnell *erfassen* wollen, mit Hilfe von kompakten aber vollständigen Listen, Tabellen und ähnlichen Darstellungsformen nebst Erklärungen in Kurzform, begleitet von Kodebeispielen.

Dieses Buch stellt darauf ab und bietet im Teil I auf etwa 40 Seiten diese Möglichkeit, wobei gleichzeitig eine Referenz gegeben ist.

Dieses Buch basiert auf einem Dokument [1], das seit Januar 2000 als Webseite im Internet existiert und dort außerordentlich beliebt ist. Es erreichten den Verfasser eMail-Zuschriften, die genau die zuvor beschriebene Kompaktheit, die Listen, die Tabellen, den daraus resultierenden schnellen Zugriff und die gleichzeitige relative Vollständigkeit lobten. Ebenso wurde die Kommandoliste zum Editor `vi` kommentiert (Anhang) [2].

Dieses papierne Buch *setzt da noch einen drauf*, indem es besser gegliedert, umfangreicher, noch sorgfältiger geschrieben und fehlerfreier ist.

[2] Dennis Ritchie prägte auch sehr wesentlich die Entwicklung von UNIX.

Der C-Standard verwendet ein Vokabular, das in einem Lernbuch nicht dienlich wäre, wie beispielsweise *Übersetzungseinheit* für *C-Datei*, *Übersetzer* für *Compiler*, *Basisausführungszeichensatz* für *Zeichensatz*, und so weiter. In diesem Buch hingegen werden allgemein übliche Begriffe verwendet, wie sie in der täglichen Praxis anzutreffen sind (► S. 203).

Im Zusammenhang mit Programmierung gibt es einige Abhängigkeiten von der verwendeten Plattform (Prozessor, Betriebssytem). In diesem Buch wird nötigenfalls von der gängigsten Plattform ausgegangen, nämlich INTEL iX86. Das ist u. a. bei konkreten Zahlenbeispielen und Bitrepräsentationen oft unvermeidbar.

Die Sprache C ist allerdings mit großem Abstand die portabelste Programmiersprache überhaupt, so daß in der Programmierpraxis mit C kaum unterschiedlich programmiert werden muß, um verschiedene Plattformen zu bedienen.

Desweiteren ist C generell wohl die am weitesten verbreitete Programmiersprache. Entwicklungssysteme für Mikrokontroller haben nahezu ausnahmslos C-Compiler. Eine andere Tendenz ist nicht in Sicht. Das wäre auch Unfug, denn für die Programmierung von Betriebssystemen, deren Kommandos und die Millionen von Programmen für Mikrokontroller ist eine eher wenig abstrahierende Sprache wie C auch am geeignetsten.

C ist wegen ihrer Feinkörnigkeit, Rekursivität, Flexibilität und Ressourcenschonung für die zuvor aufgeführten Anwendungsgebiete einfach ideal. Es können aber auch alle anderen Gebiete gut bis ausreichend mit C bedient werden, besonders wenn es auf Ressourcenschonung und maximale Effizienz ankommt. Wenn es darauf nicht ankommt, sind manche anderen Programmiersprachen komfortabler und *sicherer*.

Das Manuskript zu diesem Buch wurde hergestellt mit: dem Editor `gvim`, dem vom Verfasser selbst entwickelten Skript-Interpreter `bsh` und – LaTeX.

UNIX-Betriebssystem war FREEBSD V4.11 mit Programmpaket `teTeX-3.0`.

Die Buch-CD enthält eine `.html`-Datei mit den Kodeabschnitten des Buches in Farbe. Ebenso eine `.txt`-Datei.

Bad Salzuflen, November 2004 *Helmut O. B. Schellong*

Inhaltsverzeichnis

Teil I

Erfassung der Sprache C / Referenz

1

C-Schlüsselwörter

Schlüsselwörter sind in allen Programmiersprachen reservierte Wörter. In C sind auch alle Namen `_[A-Z]...` und `__...` reserviert, auch `_...` bereichsweise. Weiterhin durch die Standard-<Header> eingeführte Bezeichner.

1.1 Liste der Schlüsselwörter

Typen von Datenobjekten

`char`	Integer
`short`	Integer
`int`	Integer (Voreinstellung)
`long`	Integer
`float`	Gleitkomma
`double`	Gleitkomma
`void`	leer/unbestimmt
`unsigned`	Integer ohne Vorzeichen
`signed`	Integer (explizit) mit Vorzeichen
`struct`	Beliebig zusammengesetzter Typ (Struktur)
`union`	Auswahltyp/Vereinigungstyp
`enum`	Aufzählungstyp (Enumeration; Integer)
`typedef`	Individuelle Typvereinbarung

Byte-Anzahl von Daten-Objekten und -Typen

`sizeof`	Beispiele: `sizeof(int), sizeof i`

Speicherklassen von (Daten-)Objekten

`auto`	Voreinstellung in Funktionen
`register`	(Möglichst) ein Prozessor-Register verwenden
`static`	Name/Symbol wird nicht exportiert; statisch
`extern`	Objekt (public), von außen her bekanntmachen

Typqualifizierer von Daten-Objekten

`const`	Nach Initialisierung read-only
`volatile`	Keine Optimierung/Stets Direktzugriff

Programmierung des Ablaufes

`if`	Bedingte Verzweigung (falls ja)
`else`	Bedingte Verzweigung (falls nein)
`while`	Schleife
`for`	Schleife
`do`	Schleife (do-while)
`switch`	Fallunterscheidung (case)
`case`	Fallunterscheidung (Fallzweig)
`default`	Fallunterscheidung (default-case)
`break`	Hinaussprung: Schleife/switch
`continue`	Fortsetzungssprung in Schleifen
`goto`	Allgemeiner unbedingter Sprung
`return`	Beenden einer Funktion (Rücksprung)

Neuer C-Standard C99

`inline`	Funktionen Inline
`restrict`	Privater Speicherbereich für Zeiger
`_Bool`	Wertebereich `{0,1}`
`_Complex`	z = real + imaginär
`_Imaginary`	imaginär

Die Schreibweise `_Xyyy` bei den neuen Schlüsselwörtern wurde wahrscheinlich so gewählt, weil beispielsweise `bool` als Preprocessor-Definition zuvor bereits weit verbreitet war und/oder wegen des Zusammenhangs von C und C++.

1.2 Erklärung einiger besonderer Schlüsselwörter

const

```
const int ci= 16;
const int *cip= adr;
```

Hiernach sind `ci= x;` und `*cip= x;` Fehler!

```
int * const ipc= &ci;
```

Hiernach wäre `ipc++;` ein Fehler! Aber, diese Definition ist problematisch, denn `(*ipc)++;` wäre ja erlaubt, was aber `ci` ändern würde, `ci` ist jedoch konstant!

```
int const * const ccip= &ci;
```

Hiernach sind `*ccip= x;` und `ccip= a;` Fehler!

```
extern int atoi(const char *);
```

Optimierung möglich, da der Compiler davon ausgehen darf, daß die Funktion `atoi` auf das per Adresse übergebene Objekt keine Schreibzugriffe vornimmt.

volatile

```
static void seKulmination(void)
{
   int volatile rf=0;
   if (setjmp(J)==1 && rf>0)  return;
   // ...
   if (seKul>9)  rf=2, ++a;
   if (a)  longjmp(J, 1);
   ErrV= rf;  // cc
   return;
}
```

Ohne `volatile` hält der Compiler die Variable `rf` wahrscheinlich zeitweilig in einem Register und ändert sie *nicht* an der Stelle `rf=2`, sondern erst an der Stelle `cc`, weil sie bis dahin gar nicht gebraucht wird. Sie wird aber doch gebraucht, falls `longjmp()` aufgerufen wird, denn `longjmp()` kehrt nicht zurück, sondern springt nach `setjmp()`, wo `rf>0` steht. Und genau *das* kann der Compiler nicht wissen!

Mit `volatile` wird auf die Variable `rf` bei *jedem* Vorkommnis *direkt* zugegriffen. Eine weitere Fehlermöglichkeit existiert im Zusammenhang mit globalen Variablen, die laufend in Interrupt-Routinen verändert werden. (▶ S. 209)

typedef

Dieses Schlüsselwort gestattet die Definition eigener Typen. Komplexe Typen sind besonders einfach zu definieren durch mehrstufige Anwendung.

```
typedef  unsigned char  BYTE;
typedef  BYTE  *BYTEP;

BYTE a, b=2, c, *bp=0;
```

```
typedef struct _FILE_
{
   int             __cnt;      // 4
   unsigned char *__ptr,       // 8
                 *__base,      //12
                  __flag,      //13
                  __file,      //14
                  __buf[2];    //16(Dummy)
} FILE, *FILEP;

extern FILE __iob[];

#define stdin   (&__iob[0])
#define stdout  (&__iob[1])
#define stderr  (&__iob[2])

FILE *fp;
fp= fopen("COM2", "rb");
if (!fp)  PrintErr(E_OPENF, "COM2"), exit(2);
```

Hier wurden zuletzt die Typen `FILE` und `FILEP` definiert. Anschließend ein Array `__iob[]` bekannt gemacht, aus Strukturen vom `FILE`-Typ als Elemente, das außerhalb angelegt ist, weshalb es als *unvollständiger Typ* angegeben wurde (`[`ohne Angabe`]`), da die Elementanzahl in der aktuellen C-Quelle unbekannt ist. Man sieht auch, daß die vordefinierten `FILE`-Pointer `stdin`, `stdout`, `stderr`, die Adressen der ersten drei Array-Elemente sind, vom Typ `FILE*` beziehungsweise `FILEP`.
(▶ 37, 132, 210, 115)

```
typedef  int A[10][2];
A a, b;
int a[10][2], b[10][2];
```

Es wurde ein Typ `A` definiert. Die beiden letzten Zeilen haben gleiche Wirkung.

Sachverzeichnis