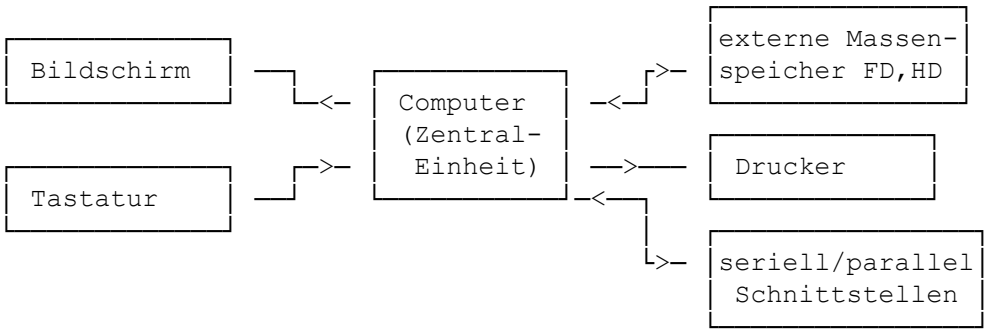
5. Die Dateioperationen unter Standard C

Eine leistungsfähige Datenverarbeitungsanlage (DV) verfügt über vielfältige Möglichkeiten der Datenein- und ausgabe. Nach dem in der EDV vorherrschenden EVA-Prinzip (Dateneingabe -> Datenverarbeitung -> Datenausgabe) findet die eigentliche Datenverarbeitung in der Zentraleinheit (oft als Prozessor bezeichnet) des Computers statt. Alle an der Zentraleinheit angeschlossenen Peripheriegeräte dienen entweder der Dateneingabe, z. B. Tastatur oder der Datenausgabe, z. B. Bildschirm, Drucker.
Eine Sonderstellung nehmen die externen Massenspeicher ein, sie gehören sowohl der Gruppe der Dateneingabe- als auch der Datenausgabegeräte an. Während im Arbeitsspeicher (RAM) die Daten nur temporär gespeichert sind und nach dem Ausschalten des Rechners unwiederbringlich verloren gehen, handelt es sich bei den externen Speichern um permanente - meist magnetische - Datenträger mit hoher Speicherkapazität.



Beabsichtigt man, die Ergebnisse seines mühevollen Schaffens der Nachwelt zu erhalten, müssen die Daten im Arbeitsspeicher auf ein externes Speichermedium ausgelagert, d.h. gespeichert bzw. gesichert werden. Unter dem Betriebssystem Windows werden die Daten immer in einer Datei abgelegt. Jede Datei hat einen Dateinamen, über den die Daten gespeichert und wieder aufgefunden, d. h. gelesen werden können.

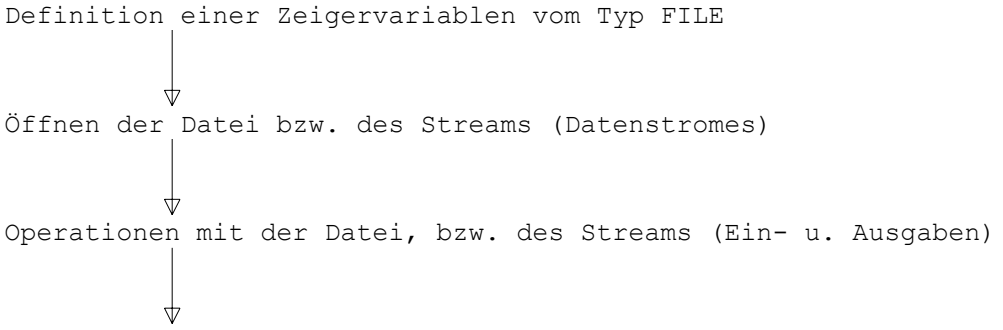
Standard C stellt aus Gründen der Portabilität keine Dateiverwaltungsbefehle zur Verfügung. Die Dateiverwaltung wird ausschließlich unter Zuhilfenahme von speziellen Ein- und Ausgabefunktionen der Standardbibliothek durchgeführt. Sie muss deshalb grundsätzlich bei Dateioperationen am Anfang des Quellprogrammes eingebunden werden.

```
#include <stdio.h>
```

In Standard C existieren zwei Arten von Dateiverwaltungsfunktionen:
- die gepufferte Dateisystem (Ein- und Ausgabe über Streams = Datenstrom) und das
- ungepufferte Dateisystem

Das ungepufferte Dateisystem entstand bei der Implementierung der C-Compiler auf Unix Systemen. Da sich das gepufferte und das ungepufferte Dateisystem recht ähnlich sind, wird nach neuen Normungsbestrebungen des ANSI Komitees in Zukunft nur noch das gepufferte Dateisystem verwendet (erweitert um die Bearbeitung von Text- und Binärdateien) und auf die ungepufferte Dateiverwaltung verzichtet. Ersteres soll nun näher betrachtet werden.

Die nachstehende Grafik verdeutlicht den Einsatz des gepufferten Dateisystems:



- Definition einer Zeigervariablen vom Typ FILE

Im Programm wird auf die Datei über einen Zeiger vom Typ FILE zugegriffen:

```
  FILE  *filevar      /* filevar ist logischer Dateiname */
```

FILE ist ein strukturierter Datentyp, der in der Headerdatei stdio.h definiert ist. Mit Hilfe des FILE-Datentyps verwaltet die Standardbibliothek die Bearbeitung einer Datei.

- **Öffnen der Datei bzw. des Streams (Datenstromes)**

Bevor Lese- und Schreiboperationen auf eine Datei ausgeführt werden können, muss sie geöffnet werden. Hierbei findet eine Zuordnung der ein- und ausgehenden Daten zu der vorher im Programm definierten Zeigervariablen (logischen Dateivariable) statt. Die Eröffnung erfolgt über die Funktion fopen(..), die gleichzeitig einen Schreib-/Lesepuffer (im Arbeitsspeicher RAM) einrichtet. Die eigentliche Datenübertragung erfolgt physikalisch jedoch erst dann, wenn der Puffer voll ist, oder explizit die Funktion fclose(..) aufgerufen wird:

```
  filevar = fopen(filename, mode)
  char  *filename            /* physikalischer Name der Datei */
  char  *mode                /* Eröffnungsmodus */
```

Der Variablen filevar wird ein Zeiger auf die Datei filename zugewiesen. Alle weiteren Dateioperationen geschehen von jetzt an über filevar.
mode ist ebenfalls ein String und legt den Eröffnungsmodus fest.

Modus	**Erklärung**
"r"	Öffnen zum Lesen
"w"	Öffnen zum Schreiben
"a"	Öffnen zum Erweitern
"r+"	Öffnen zum Schreiben und Lesen Die zu eröffnende Datei muss bereits bestehen
"w+"	Öffnen zum Schreiben und Lesen; Eine bereits bestehende Datei wird überschrieben
"a+"	Öffnen zum erweitern, Besteht die Datei noch nicht, so wird diese erst angelegt; zusätzlich können weitere Parameter angegeben werden:
"b"	eröffne als Binärdatei
"t"	eröffne als Textdatei (default-Modus)

Bei der Bearbeitung von Dateien unterscheidet Standard C nochmals zwei Modi:

Text: Die von einem Stream gelesene Zeichenfolge CR/LF (ASCII 13 und 10) werden automatisch auf ein einzelnes LF reduziert. Umgekehrt wird bei Schreibaktionen statt eines einzelnen LF die Zeichenfolge CR/LF gesetzt.

Binär: Es findet hier keine Umwandlung von LF in CR/LF und umgekehrt statt.

Beipiel:

```
filevar = fopen("ROHDATEN.DAT","r+t")
```

Die physikalische Datei ROHDATEN.DAT wird dem Zeiger filevar zugeordnet, und gleichzeitig für den Lesezugriff ("r") geöffnet; die Datei wird im folgenden wie eine Textdatei behandelt. Tritt ein Fehler bei der Dateieröffnung auf, z.B. wenn die angegebene Datei nicht existiert, dann erhält filevar den Wert NULL.

Beispiel:

```
#include <stdio.h>

main()
{char filename[25];
 FILE *filevar;

 printf("Gebe Dateiname an: ");
 scanf("%s",filename);  /* Liest den gewünschten Dateinamen */
 if ((filevar=fopen(filename,"r"))==NULL)
   printf("Fehler bei der Dateieröffnung");
 else
   printf("Die Datei %s wurde eröffnet.\n", filename);
}
```

Eine Besonderheit der Sprache Standard C ist die Handhabung der Peripheriegeräte: Jedes externe Gerät wird als Datei betrachtet, d.h. Ein- und Ausgaben in und von Dateien werden genauso behandelt wie von der Tastatur oder auf dem Bildschirm. In der Bibliothek stdio.h von Standard C sind sehr oft benötigte Streams (Standardkanäle) bereits vordefiniert. Beim Start eines Standard C Programms werden diese automatisch eröffnet.

Name	**Modus**	**Erklärung**
stdout	Text	Standardausgabe Bildschirm
stdin	Text	Standardeingabe Tastatur
stdprn	Text	Standardausgabe auf Drucker
stdaux	Binär	Standardein- und ausgabe, serielle Schnittstelle
stderr	Binär	Standardausgabe für Fehlermeldungen

- **Operationen mit der Datei, bzw. des Streams (Ein- u. Ausgaben)**

Beim Aufruf von Dateifunktionen muss immer die in fopen(..) gesetzte Zeigervariable *fp als Argument mit angegeben werden.

Übersicht der wichtigsten Dateioperationen:

```
fputc(char Zeichen, FILE *fp)     // Gibt ein Zeichen auf den Stream, falls
                                  // erfolgreich, wird Wert >0 zurückgegeben, sonst -1

fgetc(FILE *fp)                   // Liest ein Zeichen vom Stream; das
                                  // Zeichen ist vom Typ int

fputs(char *String, FILE *fp)     // übergibt Zeichenkette zum Stream;
                                  // wenn Ausgabe erfolgreich 0; sonst -1

fgets(char *String,               // Liest Zeichenkette mit Länge von Stream
      int Länge, FILE *fp)        // Rückgabe: Zeiger auf Zeichenkette

fprintf(FILE *fp,                 // wie Funktion printf(..); jedoch erfolgt
        char *Formatstring,       // formatierte Ausgabe auf Datei
        void Argumente)

fscanf(FILE *fp,                  // wie Funktion scanf(..); jedoch wird von
        char *Formatstring,       // Stream gelesen; Rückgabe ist die Anzahl
        void Argumente)           // der gelesenen Argumente

ferror(FILE *fp)  // prüft ob eine Dateioperation erfolgreich ausgeführt wurde;
                  // falls Fehler auftreten, wird ein Wert <>0 zurückgegeben,
                  // sonst wird 0 zurückgegeben. Die Funktion sollte nach jeder
                  // Dateioperation aufgerufen werden)

remove(char *filename)   // löscht eine Datei physikalisch; falls
                         // erfolgreich wird 0 zurückgegeben, sonst ein Wert <>0
```

Allgemein erfolgt der Zugriff auf eine geöffnete Datei zum Lesen bzw. Schreiben sequentiell, d. h. es werden alle Daten der Reihe nach ausgegeben. Speziell für eine formatierte Aus- und Eingabe in Textdateien stellt Standard C die Funktionen fprintf(..) bzw. fscanf(..) zur Verfügung.

Soll der Zugriff frei wählbar auf bestimmte Stellen in der Datei erfolgen (wahlfreier Zugriff: Random-Access), so kann hierzu die Position des Dateizeigers bewegt werden, ohne das gelesen oder geschrieben wird.

```
feof(FILE *fp)    // prüft, ob Ende der Datei (EOF) erreicht
                  // wenn Ende erreicht wird ein Wert <> 0
                  // zurückgegeben, sonst 0

ftell(FILE *fp)   // ermittelt die Position des Dateizeigers
                  // in der Datei

fseek(FILE *fp,            // positioniert den Dateizeiger neu
      long int Adresse     // Adresse= Anzahl der Bytes, um die der
      int mode)            // Zeiger bewegt werden soll:
                           // pos. für vorwärts, neg. für rückwärts;
                           // mode gibt die Relation an:
                           // 0 -> rel. zum Dateianfang,
                           // 1 -> rel. zur aktuellen Position
                           // 2 -> rel. zum Dateiende

rewind(FILE *fp)   // setzt die Position des Dateizeigers auf
                   // den Anfang der Datei
```

Die Funktionen fread(..) und fwrite(..) sind neu im gepufferten Dateisystem und wurden erst durch das ANSI Komitee definiert.

```
int fread(char *puffer, int groesse, int anzahl, FILE *fp);
```

Die Funktion fread(..) liest von der Datei, spezifiziert in fp, eine bestimmte Datenmenge in den Arbeitsspeicher des Computers. Die Startadresse der Daten wird durch den Pointer puffer angegeben. Die Größe der zu lesenden Datenmenge durch die Variablen anzahl und groesse vorgegeben.

Beispiel:

Es sollen vier float-Zahlen nach puffer gelesen werden. Der Funktionsaufruf muss demnach lauten:

```
fread(puffer,sizeof(float),4,fp);
```

Das Funktionsergebnis ist die Anzahl der wirklich gelesenen Daten.

Die Umkehrung, d.h. das Schreiben in eine Datei erledigt die Funktion fwrite(..). Die Angaben haben dieselbe Bedeutung wie fread(..). Zurückgegeben wird die Anzahl der wirklich geschriebenen Daten. Diese kann bei Erreichen des Dateiendes kleiner sein als die Anzahl im Funktionsaufruf.

```
int fwrite(char *puffer, int groesse, int anzahl, FILE *fp);
```

- **Schließen der Datei bzw. des Streams**

Nach der Benutzung müssen Dateien wieder geschlossen werden. Dies geschieht entweder bei der Programmbeendigung oder, wenn die maximale Zahl der gleichzeitig zu öffnenten Dateien erreicht ist (wird unter MS DOS in CONFIG.SYS festgelegt: z.B. FILES=10). Die Funktion fclose(..) leert den Datenpuffer des Streams und schließt dann die Datei. Sie gibt den Wert 0 zurück, wenn die Datei erfolgreich geschlossen werden konnte, sonst wird der Wert -1 zurückgegeben.

```
  filecount = fclose(filevar);       /* vorher: FILE *filevar */
```

Werden viele Dateien während des Programmlaufes geöffnet, so können diese allesamt mit der Funktion fcloseall(..) geschlossen werden. Die Funktion übergibt zudem die Anzahl der geschlossenen Dateien.

```
  filecount= fcloseall(filevar);
```

5.1 Beispiele zur Veranschaulichung der Dateioperationen

```
/*********************************************************************/
/* Liest eine Datei und gibt sie zeilenweise auf dem Monitor aus.    */
/* Demonstration der Funktion fgets(..)                              */
/*********************************************************************/
#include <stdio.h>
main()
{ FILE *fp;
  char zeile[80], filename[25];
  char *str;                    /* str=  Zeigervariable auf ein Zeichen */

  printf("Geben Sie den Dateinamen ein -> ");
  scanf("%s", filename);          /* liest den Dateinamen ein */
  fp=fopen(filename,"r");      /* öffnet Textdatei zum Lesen */
```

```
  do
  {
   str = fgets(zeile,80,fp);     /* holt eine Zeile der Datei */
   if (str != NULL)
      printf("%s\n",zeile);    /* stellt eine Zeile auf Monitor dar */
  } while (str != NULL);         /* wiederhole bis Dateiende (->NULL) erreicht
*/

  fclose(fp);
}



/*********************************************************************/
/* Das Programm schreibt den über Tastatur eingegebenen Text in eine  */
/* Datei. Zieldatei muss in der Kommandozeile angegeben werden,        */
/*  z.B. auto.bat.  Aufruf von MS-DOS: EDIT auto.bat                  */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
 FILE *filepointer;
 char  ch;

 if (argc != 2)     /* argc gibt die Anzahl der übergebenen Parameter an */
 {
  printf("Es wurde keine Zieldatei angegeben !");
  exit(1);            /* gibt an die MS-DOS Umgebung den Wert 1 zurück */
 }

 if((filepointer=fopen(argv[1], "w"))==NULL)  /* übergebene Datei wird geöffnet*/
 {
  printf("\nSorry, ich kann die Datei %s nicht öffnen !", argv[1]);
  exit(1);
 }
 do
 {
  ch=getchar();             /* Liest ein zeichen von Standardgerät stdin */
  putc(ch,filepointer);     /* Schreibt zeichenweise nach Stream */
 } while (ch !='$');

 fclose(filepointer);       /* Schließen der Datei */
}



/*********************************************************************/
/* Programm zum Kopieren von Dateien                                  */
/* Quell- und Zieldatei müssen in der Kommandozeile angegeben werden  /*
/* Die Übertragung erfolgt Zeichen für Zeichen                        */
/* Aufruf von MS-DOS: KOPIERE config.sys config.bak                   */
/*********************************************************************/
#include <stdio.h>
#define LESEN  "rb"           /* Binäre Datei zum Lesen */
#define SCHREIBEN  "wb"           /* Binäre Datei zum Schreiben */

main(int argc, char *argv[])
{
 FILE *in, *out;
```

```
 char  ch;

 if (argc != 3)     /* argc gibt Anzahl der übergebenen Parameter an */
 {
  printf("Übergebene Parameter sind nicht vollständig !");
  exit(1);            /* gibt an die MS-DOS Umgebung den Wert 1 zurück */
 }

 if((in=fopen(argv[1], LESEN))==NULL)   /* übergebene Datei wird geöffnet */
 {
  printf("\nSorry, ich kann die Datei %s nicht öffnen !", argv[1]);
  exit(1);
 }

 if((out=fopen(argv[2], SCHREIBEN))==NULL)  /* Übergebene Datei wird geöffnet */
 {
  printf("\nSorry, ich kann die Datei %s nicht öffnen !", argv[2]);
  exit(1);
 }


 while (!feof(in))
 {
  putc(getc(in),out);  /*liest und schreibt zeichenweise von/nach nach Stream */
 };

 fclose(in);           /* Schließen der Dateien */
 fclose(out);
}




/*****************************************************************/
/* Disketten-Utility: DUMP <filename>                            */
/* Das Programm zeigt den Inhalt einer Datei in ASCII-Darstellung     */
/* und hexadezimal an.                                           */
/* Demonstration: wahlfreier Dateizugriff mit fseek(..)          */
/*****************************************************************/
#include <stdio.h>
#include <ctype.h>      /* enthält Funktion isprint(..) */
#include <stdlib.h>     /* enthält Funktion exit(..) */
#define SEKTORSIZE  128
#define LESEN  "rb"

void Anzeige(int liessekt);      /* Prototyp der Funktion */

char  buffer[SEKTORSIZE];       /* globale Variablendeklaration */


main(int argc, char *argv[])
{
 FILE *in *out;
 int sektor, liessekt;

 if (argc != 2)     /* argc gibt Anzahl der übergebenen Parameter an */
 {
  printf("übergebene Parameter sind nicht vollständig !");
  exit(1);            /* gibt an die MS-DOS Umgebung den Wert 1 zurück */
```

```
 }

 if((fp=fopen(argv[1], LESEN))==NULL)   /* übergebene Datei wird geöffnet */
 {
  printf("\nSorry, ich kann die Datei %s nicht öffnen !", argv[1]);
  exit(1);
 }

 do
 {
  printf("Gib Sektor-Nr. : ");
  scanf("%1d", &sektor);
  if (fseek(fp,sektor*SIZE,SEEK_SET))    /* SEEK_SET = Konstante für Dateianfang
*/
      printf(" SEEK Error !\n");          /* positioniere Dateizeiger */

  if ((liessekt= fread(buffer,1,SIZE,fp)) != SIZE)
      printf(" Dateiende erreicht !\n");  /* Lesezugriff */

  Anzeige(liessekt);
 } while(sektor>=0);
}    /* Ende Hauptfunktion */


void Anzeige(int sektnr)
/*************  Zeigt einen Sektor auf dem Bildschirm an **************/
{
 int i,j;

 for (i=0; i<=sektnr/16; i++)
 {
  for (j=0; j<16; j++)  printf("%3X", buffer[i*16+j];
  printf("   ");
  for (j=0; j<16; j++)
  {
   if (isprint(buffer[i*16+j]))  printf("%c", buffer[i*16+j]);
   else print(".");
  }
  printf("\n");
 }
}


/*********************************************************************/
/* Beispiel für die Lösung des Problems, eine Textdatei (in diesem    */
/* Fall TEST_PRN.TXT) auf den Drucker umzuleiten d.h. auszudrucken.   */
/*********************************************************************/
#include <stdio.h>
main()
{
 FILE *beispiel;      /* Erklärung von Pointern */
 FILE *printer;
 char ch;

 beispiel = fopen("TEST_PRN.TXT","r");    /* Öffne Eingabe-Datei */
 printer = fopen("PRN","w");              /* Öffne Druck-Datei */

 do
 {
  ch = getc(beispiel);    /* Hole ein Zeichen von Datei TEST_PRN.TXT */
```

```
  if (ch != EOF)
   {
    putchar(ch);          /* Gib Zeichen auf Monitor aus */
    putc(c,printer);      /* Gib Zeichen auf Drucker aus */
   }
 } while (ch != EOF;      /* Wiederhole bis EOF (end of file) */

 fclose(beispiel);        /* Schließen der Dateien */
 fclose(printer);
}
```

5.2 Übungsaufgaben zu den Dateioperationen

Übung 1: Dateien vergleichen

Schreiben Sie ein Programm, das zwei Textdateien Quelldatei und Zieldatei vergleicht.
Wenn die beiden Dateien gleichen Inhalt haben soll, soll die Meldung
- Quelldatei und Zieldatei sind identisch !
herausgegeben werden. Ist der Inhalt verschieden, soll bei dem ersten unterschiedlichen Zeichen ausgegeben werden:

- Quelldatei: gelesenes Zeichen <das Zeichen> in Zeile <Zeilennr.>
- Zieldatei: gelesenes Zeichen <das Zeichen> in Zeile <Zeilennr.>

Beachten Sie, dass auch die Zeilenstruktur übereinstimmen muss.

Übung 2: Dateien kopieren

Schreiben Sie ein Programm, das mit drei Dateien gleichzeitig arbeitet,
- eine Eingabedatei (Input)
- eine Ausgabedatei (Output)
- eine Protokolldatei zur Druckausgabe (Output)

Zuerst sollen die Namen der Ein- und Ausgabedateien über Tastatur eingelesen werden. Anschließend wird Zeichen für Zeichen die Eingabedatei gelesen und in die Ausgabedatei (Zieldatei) kopiert, gleichzeitig erfolgt Protokolldruck auf Drucker.

Übung 3: Dateien lesen und auflisten

Das Programm soll einen Dateinamen über Tastatur einlesen. Der Text wird zeilenweise auf dem Monitor ausgegeben unter Angabe der absoluten Zeilennummer, die jeder Zeile voran gestellt werden soll.

Übung 4: Schülerdatei anlegen

Schreiben Sie ein Standard C-Programm, welches eine Datei Schülerdaten erzeugt. Die Datei soll für jeden Schüler einer Klasse
- den Namen des Schülers,
- bis zu zehn Noten und
- die Durchschnittsnote enthalten.

Die Anzahl der Schüler und Noten pro Schüler soll über die Datei Input eingelesen werden. Für jeden Schüler existiert eine Eingabezeile in der Datei Input und zwar zuerst der Name mit maximal 20 Zeichen, dann ein Leerzeichen und schließlich die Noten (max. 10 Noten). Die Durchschnittsnote ist nicht angegeben und muss berechnet werden. Jede Eingabezeile wird durch ein Zeilenendezeichen (eoln = end-of-line) abgeschlossen.

6 Anhang

6.1 ASCII-Tabelle

6.1.1 Nicht druckbare Zeichen

Dez	Okt	Hex	Zeichen	Code	Bemerkung
0	000	0x00	Ctrl-@	NUL	Null prompt
1	001	0x01	Ctrl-A	SOH	Start of heading
2	002	0x02	Ctrl-B	STX	Start of text
3	003	0x03	Ctrl-C	ETX	End of Text
4	004	0x04	Ctrl-D	EOT	End of transmission
5	005	0x05	Ctrl-E	ENQ	Enquiry
6	006	0x06	Ctrl-F	ACK	Acknowledge
7	007	0x07	Ctrl-G	BEL	Bell
8	010	0x08	Ctrl-H	BS	Backspace
9	011	0x09	Ctrl-I	HT	Horizontal tab
10	012	0x0A	Ctrl-J	LF	Line feed
11	013	0x0B	Ctrl-K	VT	Vertical tab
12	014	0x0C	Ctrl-L	FF	Form feed
				NP	New page
13	015	0x0D	Ctrl-M	CR	Carriage return
14	016	0x0E	Ctrl-N	SO	Shift out
15	017	0x0F	Ctrl-O	SI	Shift in
16	020	0x10	Ctrl-P	DLE	Data link escape
17	021	0x11	Ctrl-Q	DC1	X-ON
18	022	0x12	Ctrl-R	DC2	
19	023	0x13	Ctrl-S	DC3	X-Off
20	024	0x14	Ctrl-T	DC4	
21	025	0x15	Ctrl-U	NAK	No achnowledge
22	026	0x16	Ctrl-V	SYN	Synchronous idle
23	027	0x17	Ctrl-W	ETB	End transmission blocks
24	030	0x18	Ctrl-X	CAN	Cancel
25	031	0x19	Ctrl-Y	EM	End of medium
26	032	0x1A	Ctrl-Z	SUB	Substitute
27	033	0x1B	Ctrl-[	ESC	Escape
28	034	0x1C	Ctrl-\	FS	File separator
29	035	0x1D	Ctrl-]	GS	Group separator
30	036	0x1E	Ctrl-^	RS	Record separator

31	027	0x1F	Ctrl-_	US	Unit separator
127	0177	0x7F		DEL	Delete or rubout

6.1.2 Druckbare Zeichen

Dez	Okt	Hex	Zeichen	Bemerkung
32	040	0x20		Leerzeichen
33	041	0x21	!	Ausrufungszeichen
34	042	0x22	"	Anführungszeichen
35	043	0x23	#	Doppelkreuz
36	044	0x24	$	Dollarzeichen
37	045	0x25	%	Prozentzeichen
38	046	0x26	&	Kaufmännisches UND
39	047	0x27	'	Apostroph
40	050	0x28	(	Runde Klammer auf
41	051	0x29	)	Runde Klammer zu
42	052	0x2A	*	Stern
43	053	0x2B	+	Pluszeichen
44	054	0x2C	,	Komma
45	055	0x2D	-	Minuszeichen
46	056	0x2E	.	Punkt
47	057	0x2F	/	Schrägstrich (Slash)
48	060	0x30	0	
49	061	0x31	1	
50	062	0x32	2	
51	063	0x33	3	
52	064	0x34	4	
53	065	0x35	5	
54	066	0x36	6	
55	067	0x37	7	
56	070	0x38	8	
57	071	0x39	9	
58	072	0x3A	:	Doppelpunkt
59	073	0x3B	;	Semikolon
60	074	0x3C	<	Kleiner-als-Zeichen
61	075	0x3D	=	Gleichheitszeichen
62	076	0x3E	>	Größer-als-Zeichen
63	077	0x3F	?	Fragezeichen
64	0100	0x40	@	Klammeraffe ("at")

65	0101	0x41	A	
66	0102	0x42	B	
67	0103	0x43	C	
68	0104	0x44	D	
69	0105	0x45	E	
70	0106	0x46	F	
71	0107	0x47	G	
72	0110	0x48	H	
73	0111	0x49	I	
74	0112	0x4A	J	
75	0113	0x4B	K	
76	0114	0x4C	L	
77	0115	0x4D	M	
78	0116	0x4E	N	
79	0117	0x4F	O	
80	0120	0x50	P	
81	0121	0x51	Q	
82	0122	0x52	R	
83	0123	0x53	S	
84	0124	0x54	T	
85	0125	0x55	U	
86	0126	0x56	V	
87	0127	0x57	W	
88	0130	0x58	X	
89	0131	0x59	Y	
90	0132	0x5A	Z	
91	0133	0x5B	[	Eckige Klammer auf
92	0134	0x5C	\	Umgekehrter Schrägstrich (Backslash)
93	0135	0x5D	]	Eckige Klammer zu
94	0136	0x5E	^	Caret (Hut)
95	0137	0x5F	_	Unterstrich
96	0140	0x60	`	"Back quote"
97	0141	0x61	a	
98	0142	0x62	b	
99	0143	0x63	c	
100	0144	0x64	d	
101	0145	0x65	e	
102	0146	0x66	f	
103	0147	0x67	g	

104	0150	0x68	h	
105	0151	0x69	i	
106	0152	0x6A	j	
107	0153	0x6B	k	
108	0154	0x6C	l	
109	0155	0x6D	m	
110	0156	0x6E	n	
111	0157	0x6F	o	
112	0160	0x70	p	
113	0161	0x71	q	
114	0162	0x72	r	
115	0163	0x73	s	
116	0164	0x74	t	
117	0165	0x75	u	
118	0166	0x76	v	
119	0167	0x77	w	
120	0170	0x78	x	
121	0171	0x79	y	
122	0172	0x7A	z	
123	0173	0x7B	{	
124	0174	0x7C	|	
125	0175	0x7D	}	
126	0176	0x7E	~	