3.1.F C-Datentypen/Datentyp char – Ergänzungen und Bilder

3.1.F.1 ASCII-Tabelle

Nr.	Z.	Nr.	Z.	Nr.	Z.	Nr.	Z.	Nr.	Z.	Nr.	Z.	Nr.	Z.	Nr.	Z.
0		16		32	␣	48	0	64	@	80	P	96	`	112	p
1		17		33	!	49	1	65	A	81	Q	97	a	113	q
2		18		34	"	50	2	66	B	82	R	98	b	114	r
3		19		35	#	51	3	67	C	83	S	99	c	115	s
4		20		36	$	52	4	68	D	84	T	100	d	116	t
5		21		37	%	53	5	69	E	85	U	101	e	117	u
6		22		38	&	54	6	70	F	86	V	102	f	118	v
7		23		39	'	55	7	71	G	87	W	103	g	119	w
8		24		40	(	56	8	72	H	88	X	104	h	120	x
9		25		41	)	57	9	73	I	89	Y	105	i	121	y
10		26		42	*	58	:	74	J	90	Z	106	j	122	z
11		27		43	+	59	;	75	K	91	[	107	k	123	{
12		28		44	,	60	<	76	L	92	\	108	l	124	|
13		29		45	-	61	=	77	M	93	]	109	m	125	}
14		30		46	.	62	>	78	N	94	^	110	n	126	~
15		31		47	/	63	?	79	O	95	_	111	o	127	

3.1.F.2 Pufferung

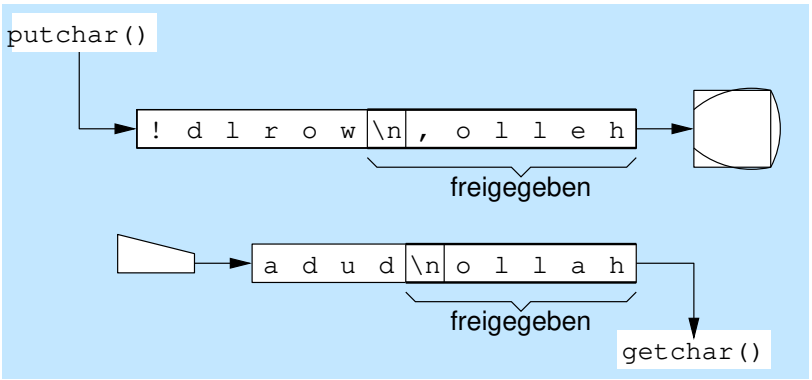

Abbildung 1: Ein- und Ausgabepufferung in Funktionen der C-Standard-Bibliothek

a) `putchar()` schreibt in den Ausgabepuffer. Nur die Zeichen, die durch ein `'\n'` freigegeben wurden, werden ausgegeben. Durch `fflush(stdout)` kann man die Ausgabe des gesamten Puffers erzwingen.

b) `getchar()` liest aus dem Eingabepuffer nur das, was durch ein `'\n'` freigegeben wurde. Falls nichts zu lesen ist, blockiert es.

   Anders ausgedrückt:
   `getchar()` liest nur, wenn im Eingabepuffer ein `'\n'` vorhanden ist. Sonst blockiert es.

3.1.F.3 Unterschied zwischen `%c` und `%d`

Was ist der Unterschied zwischen den folgenden beiden Zeilen:

```
1    printf("%c", 65);
2    printf("%d", 65);
```

Die beiden Aufrufe unterscheiden sich nur durch den Platzhalter. Oben wird `A` ausgegeben, unten `65`. Wie kommt das?

Im ersten Fall (`"%c"`) wird ausgeführt:

```
1    putchar(65); /* Ausgabe Zeichen Nr. 65, ein 'A' */
```

Im zweiten Fall (`"%d"`) wird ausgeführt:

```
1    char c;
2    c=65/10;     /* ergibt  6 */
3    c=c+48;      /* ergibt 54 */
4    putchar(c); /* Ausgabe Zeichen Nr. 54, eine '6' */
5    c=65%10;     /* ergibt  5 */
6    c=c+48;      /* ergibt 53 */
7    putchar(c); /* Ausgabe Zeichen Nr. 53, eine '5' */
```

Man könnte den Aufruf `printf("%c",x)` durch die folgende, selbst geschriebene Funktion `printc()` ersetzen:

```
1  void printc(int x)
2  {
3     putchar(x);
4  }
```

Ebenso könnte man den Aufruf `printf("%d",x)` durch diese selbst geschriebene Funktion `printd()` ersetzen (allerdings nur für Zahlen zwischen 0 und 99):

```
1  void printd(int x)
2  {
3     char c;
4     c=x/10;      /* Wert der ersten Ziffer holen */
5     c=c+48;      /* ASCII-Codenummer ermitteln   */
6     putchar(c); /* ausgeben                     */
7
8     c=x%10;      /* Wert der zweiten Ziffer holen */
9     c=c+48;      /* ASCII-Codenummer ermitteln    */
10    putchar(c); /* ausgeben                      */
11 }
```

3.1.F.4 Ein- und Ausgabe ohne Pufferung

Bei der Ausgabe von Zeichen ist die Pufferung oft lästig: Bei der Ausgabe werden Zeichen gesammelt, bis der Puffer voll ist oder ein Zeilenende erreicht wurde. Möchte man zwischendrin ein paar Zeichen ausgeben, muss die Ausgabe mit `fflush(stdout)` erzwungen werden.

Das kann man aber ändern, indem man die Pufferung um- oder abschaltet. Dazu dient die Funktion `setvbuf()`. Als Betriebsarten (einer der Parameter) sind `_IONBF` (nicht gepuffert), `_IOLBF` (Zeilenpufferung) und `_IOFBF` (voll gepuffert) möglich. Eine Kurzform davon ist die Funktion `setbuf()` mit diesem Prototyp:

```
1  int setbuf(FILE *dateimerker, char *puffer);
```

Wenn man hier für den zweiten Parameter 0 eingibt, wird dadurch die Betriebsart `_IONBF` (nicht gepuffert) eingeschaltet:

```
1    setbuf(stdout, 0); // Keine Pufferung der Ausgabe
```

In `setbuf1.c` sieht man eine Anwendung:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  int main(void)
4  {
5      setbuf(stdout, NULL);
6      while(1)
7      {
8          printf("x");
9          sleep(1);
10     }
11 }
```

Bei der Eingabe von Zeichen über die Tastatur arbeitet das Programm normalerweise erst dann weiter, wenn die ↵ -Taste gedrückt wurde, so dass ein Zeilenumbruch im Eingabepuffer der Standard-Bibliothek steht. Auch hier gibt es Abhilfe: Bei der Eingabe von der Tastatur wird `setbuf()` genauso angewandt (mit `stdin` als erstem Parameter). Allerdings muss man berücksichtigen, dass auch der Terminaltreiber für die Tastatur so arbeitet, dass er erst nach Drücken der ↵ -Taste seine Daten weitergibt. Also muss der Terminaltreiber mit einem speziellen Befehl in den Modus `raw` umgeschaltet werden. Dazu kann der Konsolenbefehl `stty` dienen. In `setbuf2.c` sieht man, wie es funktioniert:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void)
4  {
5      int ch;
6      printf("Ende␣mit␣'Q':\n");
7      system("stty␣raw"); // unter Linux: Terminal roh
8      setbuf(stdin, NULL);// C-Pufferung aus
9      ch=getchar();
10     while(ch!='Q')
11     {
12         putchar(ch);
13         fflush(stdout);
14         ch=getchar();
15     }
16     system("stty␣cooked");// unter Linux: Terminal normal
17     putchar('\n');
18     return 0;
19 }
```

Nach der Eingabe von Zeichen wird der Terminaltreiber wieder in den normalen Modus `cooked` zurückgeschaltet.

Übrigens kann man den Terminaltreiber auch dazu bringen, nicht mehr jedes eingegebene Zeichen zu wiederholen (sehr sinnvoll bei der Passworteingabe). Dazu dient der Befehl `stty -echo`. In `setbuf3.c` ist wieder ein Beispiel zu sehen:

```
1  #include <stdio.h>
2  #include <stdlib.h>
```

```c
int main(void)
{
    int ch;
    printf("Ende mit 'Q':\n");
    system("stty raw -echo"); // Terminal roh + kein Echo
    setbuf(stdin, NULL);      // keine Pufferung in C
    ch=getchar();
    while(ch!='Q')
    {
        putchar('*');
        fflush(stdout);
        ch=getchar();
    }
    system("stty echo cooked");
    putchar('\n');
    return 0;
}
```

Mit einem speziellen Systemaufruf (`ioctl()`) kann man ermitteln, ob ein Zeichen im Eingabepuffer steht. Auf diese Art kann es hinbekommen, dass `getchar()` erst dann aufgerufen wird, wenn ein Zeichen eingegeben wurde. Vorher kann man das Programm etwas anderes machen lassen. `setbuf4.c` zeigt ein Beispiel:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
int main(void)
{
    int ch ='x', anzahl;
    printf("Ende mit 'Q':\n");
    setbuf(stdin, NULL);
    system("stty raw -echo");
    while(ch!='Q')
    {
        ioctl(0, FIONREAD, &anzahl); // Anzahl Z. im Puffer ermitteln
        if (anzahl>0)
        {
            ch=getchar();
            putchar(ch);
            fflush(stdout);
        }
        else
        {
            putchar('*');
            fflush(stdout);
            sleep(1);
        }
    }
    system("stty echo cooked");
    putchar('\n');
    return 0;
}
```