Übung 5

Erarbeitet von: M.Eng. Michael Finsterbusch
Modulverantwortlicher: Prof. Dr. rer. nat. Matthias Krause
Stand: 3. Oktober 2017

Ziel der Übung ist das Kennenlernen und Vertiefen von:

- Binäroperationen
- File I/O
- Arrays
- Schleifen
- Kontrollstrukturen
- formatierte Ausgabe

Abgabe:
- `http://praktomat.hft-leipzig.de` unter „Tutorial: DKMI/DAI-17 C-Progr.“
- sämtliche Abgabemodalitäten sind im Praktomat hinterlegt

Aufgaben

Gegeben ist die Header-Datei *uebung5.h*, in der Funktionsdeklarationen enthalten sind, sowie die Datei *uebung5.c*, in der die Funktionen implementiert werden. Kopieren Sie diese in Ihr Arbeitsverzeichnis. Die Funktionen sollen entsprechen den folgenden Vorgaben implementiert werden. Um die korrekte Funktionsweise Ihrer Implementation zu testen, verwenden Sie die Funktionen in der `main()`-Funktion, die in der Datei *main.c* implementiert werden soll.

1. Implementieren Sie die Funktion `bits_print()` in der Datei *uebung5.c*. Diese Funktion soll den Wert einer gegebenen Integer-Variable (`int`) haxadezimal und binär auf der Konsole ausgeben. Beispiel:

```
1 int x=0xF172;
2 bits_print(x);
3 //Ausgabe: 0x0000F172: 0000 0000 0000 0000 1111 0001 0111 0010
```

   Hinweis: Diese Funktion können Sie zum Testen der nachfolgenden Funktionen verwenden.

2. Implementieren Sie die Funktion `bits_ror()` in der Datei *uebung5.c*. Diese Funktion erhält als Parameter einen Integer-Wert (`int`). Dieser soll um eine Position bitweise nach rechts rotiert werden. Der Unterschied zu einem Shift (`>>`) ist, dass alle Bits die nach rechts hinausgeschoben werden, wieder auf der linken Seite angefügt werden. Dies ist in Abbildung 1 dargestellt.

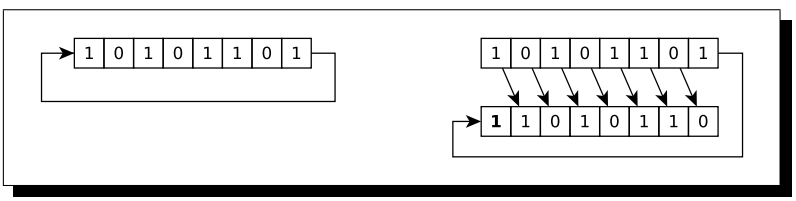

Abbildung 1: Links: schematische Darstellung der Roll-Right (ror) Funktion. Rechts: Rotation um eine Position.

3. Implementieren Sie die Funktion `bits_rol()`. Diese Funktion führt eine bitweise Rotation nach links aus. Dies ist in Abbildung 2 dargestellt.

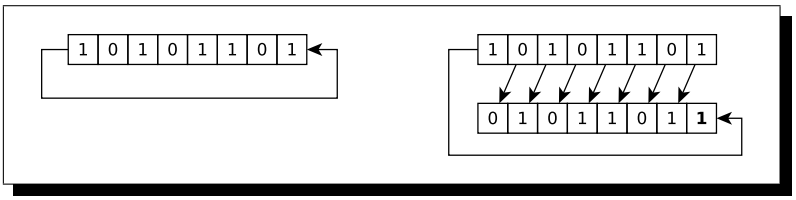

Abbildung 2: Links: schematische Darstellung der Roll-Left (rol) Funktion. Rechts: Rotation um eine Position.

4. Implementieren Sie die Funktion `bit_get()`. Mit dieser Funktion soll der Wert eines Bits, Null oder Eins, einer gegebenen Integer-Variable zurückgegeben werden. Ist der Index des Bits zu groß, also größer als `sizeof(int)·8`, soll der Rückgabewert `-1` sein.

5. Implementieren Sie die Funktion `bit_get_range()`. Mit dieser Funktion sollen mehrere Bits einer gegebenen Integer-Variable zurückgegeben werden. Sind die angeforderten Indizes der Bits zu groß, soll der Rückgabewert `-1` sein. Beispiel:

```
1  int x=0xF172;
2  int ret;
3
4  bits_print(x);
5  //Ausgabe: 0x0000F172: 0000 0000 0000 0000 1111 0001 0111 0010
6  // Bits  :            31                                7  4 3  0
7  // 3 Bit ab Position 4:                                  ---
8
9  ret=bit_get_range(x,4,3);  // Ab Position 4; 3 Bit
   ➥zurueckgeben
10 bits_print(ret);
11 //Ausgabe: 0x00000007: 0000 0000 0000 0000 0000 0000 0000 0111
```

6. Implementieren Sie die Funktion `init_lookup_table()`. In der Funktion soll das globale Array `char g_lookup_table[64]` entsprechend der Tabelle 1 initialisiert werden. Verwenden Sie dafür je eine Schleife zum befüllen des Arrays mit Klein- und Großbuchstaben sowie den Ziffern. *Hinweis:*

```
1  char c;
2  c='a'+1;
3  printf("%c %c %c\n",c,'a'+2,'A'+3); // Ausgabe: b c D
```

7. Implementieren Sie die Funktion `encode()`, mit der bis zu drei Byte kodiert werden. Die Funktion entnimmt je sechs Bits von links nach rechts aus dem Character Array. Die sechs Bit repräsentieren eine Zahl die Werte von 0 bis 63 ($2^6 - 1$) annehmen kann. Diese wird als Index für die Lookup Table aus Aufgabe 6 verwendet, um die sechs Bit in einem Byte zu kodieren, so wie es in Abbildung 3 dargestellt ist.

Tabelle 1: Werte der Lookup Table `g_lookup_table`

Index	Zeichen	Index	Zeichen	Index	Zeichen	Inedx	Zeichen
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

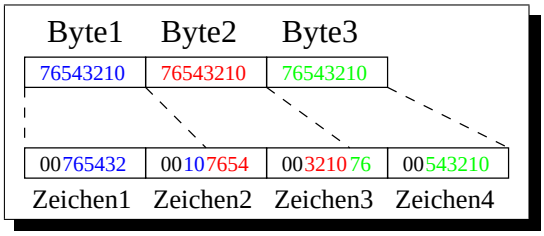

Abbildung 3: Kodierung von drei 8 Bit Zeichen in vier Symbole die nur 6 Bit nutzen. Die beiden höstwertigen Bit sind immer auf Null gesetzt.

Abbildung 4 zeigt beispielhaft, wie die Zeichenkette „ABC“ kodiert wird. Durch die Zerlegung in jeweils 6 Bits ergeben sich die Indizes 16, 20, 9 und 3. Wenn man diese in der Lookup Table nachschlägt, findet man die Zeichen „QUJD“.

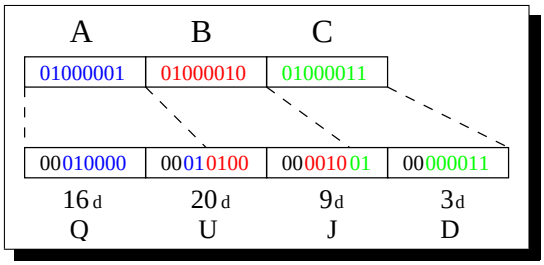

Abbildung 4: Beispiel für die Kodierung der Zeichen „ABC“

Müssen nur zwei Byte kodiert werden, bedeutet dies, dass die 16 Bit in zwei 6 Bit und einen 4 Bit Block zerlegt werden. Die fehlenden zwei Bit des 4 Bit Blocks werden mit Null-Bits aufgefüllt. Die ersten drei Ausgabe-Byte werden entsprechend der resultierenden Indizes kodiert. Das letzte Ausgabe-Byte wird mit einem Gleichheitszeichen '=' markiert (siehe Abbildung 5).

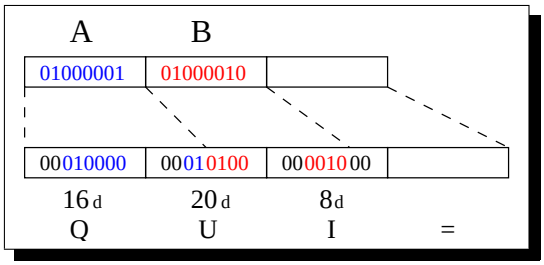

Abbildung 5: Beispiel für die Kodierung der zwei Zeichen „AB“

Wird nur ein Byte kodiert, ergeben sich ein 6 Bit und ein 2 Bit Ausgebe-Byte, an die sich zwei Geichheitszeichen '==' anschließen. Wird also das Zeichen „A“ kodiert, so ergibt sich „QQ==“.

Das folgende Listing zeigt beispielhaft, wie die Indizes berechnet und die Ausgabe-Bytes mit Hilfe der Lookup Table gesetzt werden können. Der Parameter `length` gibt die Anzahl der Eingabe-Bytes an.

```
1  int encode(char input[3],char output[4],int length)
2  {
3    unsigned char index;
4    [...]
5    index=(input[0] >> 2); // die vorderen 6 Bit des ersten Byte
6    output[0]=g_lookup_table[index];
7
8    if(length==1)
9    {
10     output[1]=...
11     output[2]='=';
12     output[3]='=';
13     return 0;
14   }
15   [...]
```

Die Rückgabewerte der Funktion `encode()` sind wie folgt definiert:

- **0** Bei Erfolg
- **1** Ungültige Länge (`length`)

Achtung: die Funktion `init_lookup_table()` muss vor der Verwendung von `encode()` aufgerufen werden!

8. In der Funktion `encode_file()` soll eine Datei geöffnet, ausgelesen, mit der Funktion `encode()` kodiert und anschließend in eine zweite Datei ausgegeben bzw. gespeichert werden. In dem Programmablaufplan in Abbildung 6 ist der logische Ablauf der Funktion dargestellt.

   Die Rückgabewerte der Funktion `encode_file()` sind wie folgt definiert:

   - **0** Bei Erfolg
   - **1** Fehler beim Öffnen der Eingabedatei
   - **2** Fehler beim Anlegen und Öffnen der Ausgabedatei

3 Fehler beim Lesen der Eingabedatei

4 Fehler beim Schreiben der Ausgabedatei

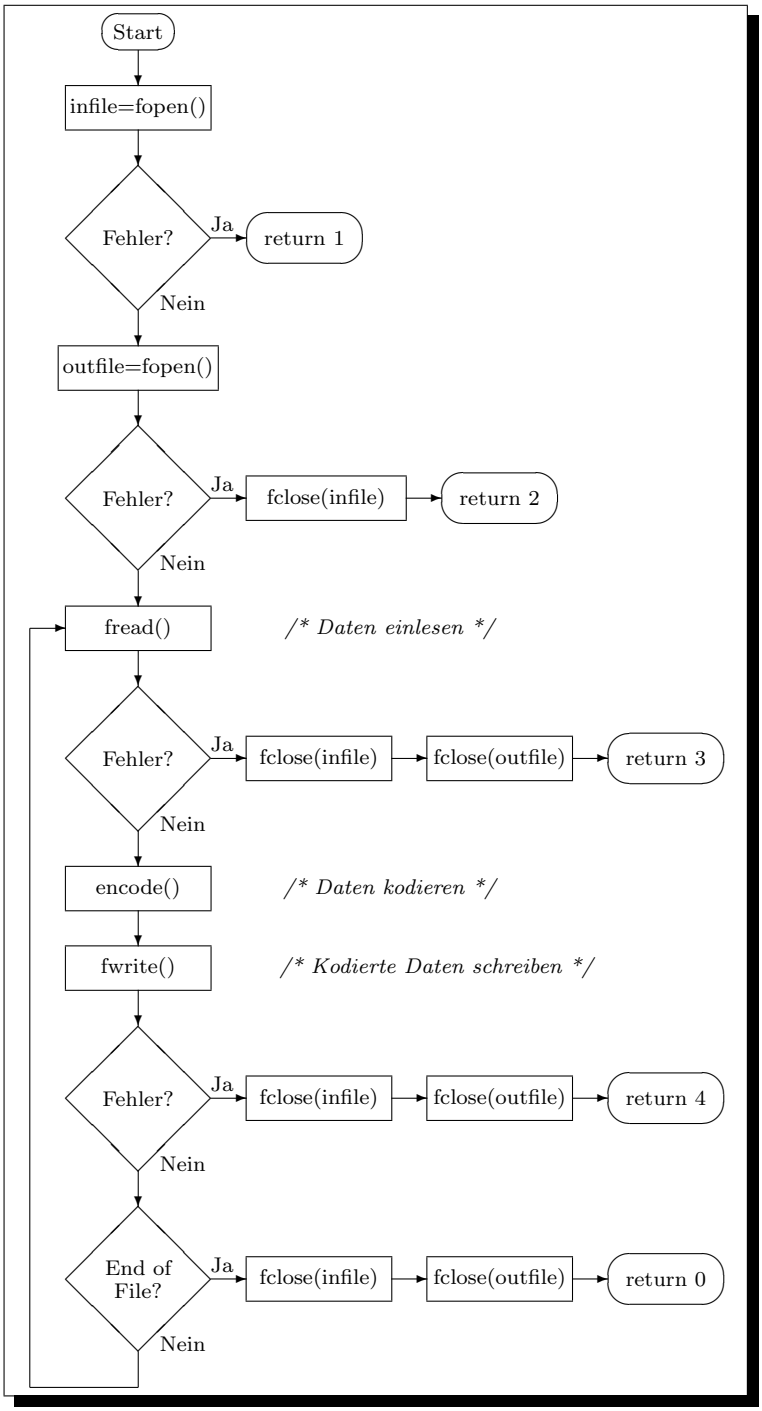

Abbildung 6: Programmablaufplan der Funktion `encode_file()`