Prof. Dr. Wolfgang Effelsberg
Dirk Stegemann Mannheim, den 13.06.2005

Vorlesung **Programmierkurs 2**, Sommersemester 2005

Übungsblatt 9

Abgabe spätestens am Montag, 20. Juni 2005, um 13.45 Uhr.

Aufgabe 1: Generischer Binärbaum

(8 Punkte, Abgabedatei `genbaum.c`)

In einem Binärbaum hat jeder Knoten außer der Wurzel genau eine eingehende Kante und höchstens zwei ausgehende Kanten, die zu seinem linken Sohn bzw. seinem rechten Sohn führen. Für jeden Knoten des Binärbaums gilt, dass sein Inhalt größer oder gleich dem Inhalt seines linken Sohns und kleiner als der Inhalt seines rechten Sohns ist, falls diese Knoten existieren.

Implementiere analog zur Aufgabe 2 auf Übungsblatt 8 einen generischen Binärbaum. Wir definieren die Datenstruktur für die Knoten wie folgt:

```
typedef struct node{
   void* inhalt;
   struct node* linkerSohn;
   struct node* rechterSohn;
} knoten;
```

Implementiere die folgenden Funktionen:

- `knoten* neuerKnoten(void* inhalt, knoten* linkerSohn, knoten* rechterSohn)` erzeugt einen neuen Knoten mit dem linken Sohn `linkerSohn`, dem rechten Sohn `rechterSohn` und mit dem durch `inhalt` referenzierten Inhalt. Die Funktion gibt `NULL` zurück, wenn der Knoten nicht erzeugt werden konnte.
- `knoten* neuerBaum()` erzeugt einen neuen Binärbaum und gibt einen Zeiger auf die Wurzel des Baums zurück.
- `knoten* einfuegen(knoten* baum, void* inhalt, int (*comp)(void* inhalt1, void* inhalt2))` fügt einen neuen Knoten mit dem übergebenen `inhalt` in den Binärbaum `baum` ein und gibt den Knoten zurück. Die Funktion gibt `NULL` zurück, wenn der neue Knoten nicht erzeugt werden konnte. `comp` vergleicht die durch die beiden Zeiger referenzierten Knoteninhalte und gibt einen Wert < 0 zurück, wenn der erste Knoteninhalt kleiner ist als der zweite, 0 wenn beide gleich sind, und einen Wert größer als 0 sonst.
- `void listeAufsteigend(knoten* baum, void (*print)(void* inhalt))` gibt die Inhalte der im Binärbaum `baum` gespeicherten Knoten mit Hilfe der Funktion `print` aufsteigend sortiert aus.
- `void baumFreigeben(knoten* baum)` gibt rekursiv den vom Binärbaum mit der Wurzel `baum` belegten Speicherplatz wieder frei.

Teste deine Implementation, indem du analog zur Aufgabe 2 auf Übungsblatt 8 eine `main`-Funktion schreibst, die vom Benutzer 5 Vornamen mit jeweils höchstens 15 Zeichen einliest und diese Namen mit Hilfe eines Binärbaums lexikografisch aufsteigend sortiert ausgibt.

Aufgabe 2: BASE64-Encoding

(8 Punkte, Abgabedatei `mime.c`)

In dieser Aufgabe betrachten wir die BASE64-Codierung, die z.B. für beliebige Binärdateien in den *Multipurpose Internet Mail Extensions* verwendet wird. Hierbei zerlegen wir eine Eingabedatei in Blöcke b der Größe 3 Bytes, d.h. $b = (b_{23}, \ldots, b_0) \in \{0,1\}^{24}$. Einen solchen Block b unterteilen wir in 4 Subblöcke $b_0^s, \ldots, b_3^s \in \{0,1\}^6$ durch $(\underbrace{b_{23}, \ldots, b_{18}}_{b_3^s}, \underbrace{b_{17}, \ldots, b_{12}}_{b_2^s}, \underbrace{b_{11}, \ldots, b_6}_{b_1^s}, \underbrace{b_5, \ldots, b_0}_{b_0^s})$.

Jeden Subblock b_j^s codieren wir anhand seiner Dezimaldarstellung mit Hilfe eines Buchstaben $c(b_j^s)$ gemäß Tabelle 1. Spätestens alle 76 Buchstaben wird der Zeichenstrom in der codierten Datei durch einen Zeilenumbruch unterbrochen.

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

Tabelle 1: Zuordnung von 6-Bit Eingaben zu Buchstaben

Für den Fall, dass die Bytelänge der zu konvertierenden Datei nicht durch 3 teilbar ist, gelten die folgenden Paddingregeln:

(i) Der letzte Block b ist 2 Bytes lang. Wir fassen b auf als $\tilde{b} = (b_{23}, \ldots, b_8, 0, , \ldots, 0) \in \{0,1\}^{24}$ und codieren b als $(c(\tilde{b}_3^s), c(\tilde{b}_2^s), c(\tilde{b}_1^s), c(pad))$.

(ii) Der letzte Block b ist 1 Byte lang. Wir fassen b auf als $\tilde{b} = (b_{23}, \ldots, b_{16}, 0, \ldots, 0) \in \{0,1\}^{24}$ und codieren b als $(c(\tilde{b}_3^s), c(\tilde{b}_2^s), c(pad), c(pad))$.

Schreibe ein C-Programm, dem über die Kommandozeile der Modus `enc` oder `dec`, der Name der Eingabedatei und der Name der Ausgabedatei übergeben werden. Im `enc`-Modus sollen die Eingabedatei nach BASE64 konvertiert und das Ergebnis in die Ausgabedatei geschrieben werden. Im `dec`-Modus soll das Programm davon ausgehen, dass die Eingabedatei in korrektem BASE64-Format vorliegt, und die Datei in Binärformat zurückverwandeln.

Hinweise: Wenn du die `main`-Funktion als `int main(int argc, char* argv[])` deklarierst, kannst du über `argc` auf die Anzahl der über die Kommandozeile übergebenen Paramter zugreifen. In `argv[i]` steht dann der i-te übergebene Parameter, wobei sich in `argv[0]` stets der Programmname befindet. Du kannst `sizeof(char)=1` annehmen und dass Großbuchstaben, Kleinbuchstaben und Zahlen jeweils zusammenhängend im Zeichensatz der Maschine angeordnet sind.

Aufgabe 3: MSP430 Assembler

(4 Punkte, Abgabe nur schriftlich)

Mache dich anhand der Erläuterungen auf den folgenden Seiten mit der Programmierung des MSP430-Sensorknotens vertraut und gib an, was das Beispielprogramm `userapp.s` leistet.

Benutzung der MSP430-Sensorknoten im π-Pool

Damit die Assembler-Aufgaben nicht nur mit Papier und Bleistift gelöst werden müssen, stehen im π-Pool mehrere Sensorboards zur Verfügung, die mit dem MSP430-Prozessor bestückt sind.

Schritt 1: Download des Testverzeichnisses

Auf den Übungswebseiten befindet sich die Datei `msptest.tar`. Diese Datei muss heruntergeladen und mit dem Befehl `tar xf msptest.tar` entpackt werden. Es entsteht ein Verzeichnis `msptest`, das ein Beispielprogramm enthält sowie ein Skript, mit dem Programme auf die Knoten überspielt werden können (mehr dazu unter Schritt 3).

Schritt 2: Anmeldung und Ausleihe

Da die Sensorknoten und die zugehörigen Kabel teurer sind, als sie aussehen, und wir auch nur wenige Exemplare davon haben, müssen wir bei der Entleihe leider etwas Bürokratie walten lassen.

Bei der ersten Ausleihe: Wer zum ersten Mal einen Knoten ausleiht, bringt bitte seinen Personalausweis mit (Studentenausweis genügt nicht). Die Aufsicht trägt seinen Namen und seine Anschrift in eine Liste ein und fügt den Account zu den entsprechenden Gruppen hinzu.

Bei jeder Ausleihe: Bei jeder Ausleihe (also auch bei der ersten) wird in eine zweite Liste der Name des Entleihers sowie der Entleihzeitpunkt eingetragen und unterschrieben. Die Rückgabe wird von der Aufsicht quittiert. Dabei ist darauf zu achten, dass der Knoten ausgeschaltet ist!

Schritt 3: Anschließen des Sensorknotens

Die Nutzung der Knoten ist nur an bestimmten Rechnern im π-Pool möglich; die Aufsicht sagt euch, welche das sind. Das Anschließen der beiden Kabel an den Knoten sollte kein Problem sein, danach muss der Sensorknoten mit dem roten Schiebeschalter eingeschaltet werden.

Start der parallelen Verbindung

Öffne eine neue Konsole, wechsle in das Verzeichnis `msptest` und führe den Befehl `make proxy` aus. Falls jetzt die Meldung

```
error: msp430: Could not find device (or device not supported)
```

kommt, dann ist der Knoten nicht eingeschaltet. Andernfalls sollte eine Meldung der Art

```
info: msp430: Target device is a 'MSP430F149' (type 7)
notice: msp430-gdbproxy: waiting on TCP port 2000
```

erscheinen. Ist dies der Fall, so ist die parallele Verbindung etabliert. Über diese Verbindung wird gleich das Programm auf den Knoten überspielt.

Start der seriellen Verbindung

Öffne eine weitere Konsole und starte das Programm `minicom`. Diese Konsole zeigt ab jetzt an, welche Daten über die serielle Schnittstelle vom Knoten an den PC übertragen werden.

Schritt 4: Compilieren und Ausführen des Programms

Jetzt bleibt noch, das Programm `userapp.s` zu compilieren und auf dem Sensorknoten auszuführen. Dazu muss zunächst der Befehl `make` ausgeführt werden. Dadurch werden eine Reihe von Dateien im Verzeichnis `out` erzeugt, unter denen sich auch die Hex-Datei `out.hex` befindet. Mit dem Befehl `make flash` wird der Inhalt dieser Datei nun auf den Knoten übertragen. Allerdings treten dabei manchmal Übertragungsfehler auf, die man an der Fehlermeldung

```
MSP430 error: Could not preserve/restore device memory (12)
```

erkennt. Beende in diesem Fall das aktuelle Programm (mit `quit`) und warte ein paar Sekunden, bis die Meldung

```
info: msp430: Target device is a 'MSP430F149' (type 7)
notice: msp430-gdbproxy: waiting on TCP port 2000
```

wieder erscheint. Unter Umständen musst du hierzu auch den Reset-Taster des Sensorknotens (den Taster bei den Schnittstellen auf der Seite der Antenne) drücken. Starte anschließend `make flash` erneut. Nach dem Flashen wird das Programm sofort ausgeführt, allerdings nicht immer ganz korrekt. Du solltest daher zunächst alle Outputs ignorieren und den Knoten mit dem Reset-Taster zurücksetzen. Nun kann die Ausgabe des Programms auf dem Sensorknoten und über den minicom beobachtet werden.

Denke bitte daran, den Sensorknoten nach getaner Arbeit wieder auszuschalten, andernfalls werden die Batterien zu schnell verbraucht!

Eigene Assembler-Programme

Zum Schreiben eigener Programme kannst du in der Datei `msptest/userapp.s` den Bereich zwischen den Kommentaren „Beginn des Hauptprogramms“ und „Ende des Hauptprogramms“ überschreiben. Die übrigen Teile der Datei sollten nicht ohne gutes Verständnis dessen, was man da tut, verändert werden. Wichtig ist auch, dass Dateiname und -verzeichnis unverändert bleiben, andernfalls muss das Makefile entsprechend angepasst werden. Genauere Hinweise zum Erstellen von Programmen und zur Ansteuerung der Sensoren gibt es ab dem nächsten Übungsblatt.