Praktikum Softprocessor SP4: Felder, Zeiger und Bitverarbeitung

7. Juli 2014

Zusammenfassung

In diesem Praktikumsversuch werden Felder, Zeiger und die Bitverarbeitung behandelt.

1 Felder (Arrays)

Ein Feld ist eine Zusammenfassung gleichartiger Elemente. Die Vereinbarung einer Feldvariablen erfolgt wie bei einem Einzelelement, nur dass nach dem Variablennamen zusätzlich die Elementeanzahl angegeben wird:

```
char str[10]; // Feld aus 10 vorzeichenbehafteten 8-Bit-Zahlen
int Liste[20];// Feld aus 20 vorzeichenbehafteten 32-Bit-Zahlen
```

Der Compiler reserviert für Felder einen zusammenhängenden Speicherbereich. Die einzelnen Feldelemente können über einen Index adressiert werden (Abb. 1). Die Elementeadresse berechnet sich stets stets wie folgt:

$$a\,(i) = a\,(0) + k \cdot i$$

(i – Index; $a\,(i)$ – Adresse von Element i; k – Byteanzahl eines Feldelementes).

Eine Zeichenkette wird in Feldern von Zeichen (8-bit-Zahlen) gespeichert. Hinter dem letzten Zeichen muss als Endkennung der Zeichenwert »0« stehen. Bei der Arbeit mit Zeichenketten ist immer darauf zu achten, dass die darzustellenden Zeichenkettenwerte kürzer als die Felder, in denen sie gespeichert werden, sind. Denn im anderen Fall überschreiben bzw. lesen Zeichenkettenoperationen auch Bytes hinter dem Zeichenfeld, die möglicherweise andere Informationen speichern.

Aufgabe 4.1: Zeichenkette spiegeln

Schreiben sie ein Hauptprogramm, in dem eine Zeichenvariable der Länge 100 vereinbart ist, das in der üblichen Endlosschleife immer

- auf die Eingabe einer mit einem Zeilenumbruch abgeschlossen Zeile wartet,

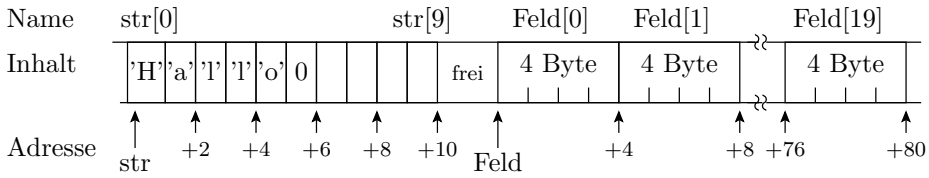

Abbildung 1: Anordnung von Feldern im Speicher

- alle Zeichen zum einen wie üblich als Echo zurück an den PC schickt und zum anderen hintereinander in die Zeichenkettenvariable schreibt,
- bei einem Zeilenumbruch die Zeichenkette mit einer echten Null abschließt[1] und
- die Zeichenkettenlänge gefolgt von einem Doppelpunkt und dem gespeicherten Inhalt in umgekehrter Reihenfolge zurück an den PC sendet.

Testbeispiel:

Eingabe:	`123␣Hallo␣Welt!␣` ↩
Ausgabe:	`123␣Hallo␣Welt!␣` `16:␣!tleW␣ollaH␣321`

Aufgabe 4.2: Feld sortieren

Schreiben sie ein Hauptprogramm, in dem ein Zahlenfeld für vorzeichenfreie 16-Bit-Zahlen mit 20 Elementen vereinbart wird und das in der üblichen Endlosschleife immer

- auf die Eingabe von maximal 20 durch Leerzeichen getrennte Ziffernfolgen zur Darstellung von Zahlen im Bereich von 0 bis $2^{16} - 1 = 65535$ gefolgt von einem Zeilenumbruch wartet
- die Zahlen mit der Funktion »term_read_unsigned« aus der Datei »utils.c« aus der vergangenen Übung in Zahlenwerte umwandelt und nacheinander in das Feld speichert
- die im Feld gespeicherten Zahlen aufsteigend mit einem Bubble-Sort sortiert und
- die sortierten Zahlen nacheinander mit der Funktion »term_write_unsigned« der Datei »utils.c« in Ziffernfolgen konvertiert und getrennt durch Leerzeichen an den PC zurücksendet.

[1] Die Zeichen für den abschließenden Zeilenumbruch sollen nicht in die Zeichenkette kopiert werden.

Testbeispiel:

Eingabe:	123␣11␣17␣122␣87␣12↩
Ausgabe:	11␣12␣17␣87␣122␣123

Hinweise:

- Es sei erlaubt, dass Sie ihre Funktion »term_read_unsigned« aus »utils.h« so abändern, dass sie die eingegebene Ziffernfolge als Echo zurückschickt.
- Sie benötigen zusätzlich Zählvariablen, z.B. eine, in der Sie die Anzahl der gespeicherten Zahlenwerte mitzählen.
- Ein Bubble-Sort besteht aus zwei verschachtelten Schleifen. Die innerste Schleife läuft immer vom ersten bis zum vorletzten Feldelement, liest das ausgewählte und das Folgelement, vergleicht beide und vertauscht sie, falls das Folgeelement kleiner als das aktuelle ist. Die äußere Schleife wiederholt den inneren Schleifenablauf solange, bis nichts mehr getauscht wird. Abbildung 2 zeigt den Sortierablauf für das Testbeispiel.

2 Zeiger

Variablen, Konstanten, aber auch Funktionen sind alles Datenobjekte, die im Speicher ab einer bestimmten Adresse angeordnet sind. Ein Zeiger ist eine Variable, die eine Adresse aufnehmen kann. Bei der Vereinbahrung wird für die Werte, auf die der Zeiger zeigen darf, ein bestimmter Typ vereinbart, z.B. »int«:

	123	↔	11						
	11		123	↔	17				
			17		123	↔	122		
					122		123	↔	78
							78	↔	12
Bubble 1: 5× tauschen									123
	11	→	17	→	122	↔	78		
					78		122	↔	12
Bubble 2: 2× tauschen							12	→	123
	11	→|	17	→	78	↔	12		
Bubble 3: 1× tauschen					12		78	→	122 → 123
	11	→	17	↔	12				
Bubble 4: 1× tauschen			12		17	→	78	→	122 → 123
	11	→	12	→	17	→	78	→	122 → 123
Bubble 5: kein Tausch, Abbruch äußere Schleife									

Abbildung 2: Sortierablauf für das Testbeispiel

```
int *ptr, *ptr1;
```

(* – Operator für »Wert eines Zeigers«). Die Adresse (der Zeigerwert) einer Variablen wird mit dem Operator »&« gebildet. Zeiger werden für die effiziente Programmierung bei der Arbeit mit Feldern verwendet. Nachfolgend wird ein Feld definiert und dem Zeiger die Adresse des ersten Feldelements zugeweisen:

```
int Feld[20];
ptr = &Feld[0]; // identisch mit ptr = Feld
```

Der Feldname ohne Indexangabe ist identisch mit dem Zeigerwert auf den Feldanfang. Zur Berechnung der Adresse andere Feldelemente wird der Indexwert zur Adresse des Feldanfangs addiert:

```
ptr1 = ptr + 4; // Addition von 4×Elementgröße
```

Im Beispiel sind die Elemente vom Typ »int« und damit vier Byte groß. Die Adresse wird um 16 (Bytespeicherplätze) vergrößert (Abb. 1). Den Wert von Element vier erhält man wahlweise mit den Ausdrücken »&(ptr + 4)«, »&(Feld+4)« oder »Feld[4]«.

Zeichenkettenvariablen sind Felder mit Zeichen (char) als Elemente. Die Werte werden als Folge von Zeichenwerten mit einer echten Null als Endekennung gespeichert (Abb. 1). Schleifen vom Typ »Wiederhole für jedes Zeichen« werden als Schleife »Wiederhole, bis der Zeichenwert null ist« programmiert. Im nachfolgenden Beispiel werden eine Zeichenkettenkonstante »Text«, eine Zeichenkettenvariable »Feld« und zwei Zeiger vereinbart. Die beiden Zeiger erhalten die Anfangsadressen der beiden Felder. Dann wird in einer Schleife, bis der Zeiger auf »Text« auf ein Element mit dem Wert »0« zeigt, der Wert auf dasselbe Feldelement der Zeichenkettenvariablen kopiert. Nach der Schleife wird zusätzlich die abschließende Null an die kopierte Zeichenfolge in der Zeichenkettenvariablen angehängt:

```
char Text = "Hallo";
char Feld[20], *dest, *src;
src  = Text;
dest = Feld;
while(*src != 0)
{
 *dest = *src;
 src++;
 dest++;
}
*dest = 0;
```

Die Parameterübergabe an Unterprogramme erfolgte bisher immer als nur lesbare »Werte«. Die übergebenen Werte werden dazu in lokale Variablen des Unterprogramms kopiert. Wertzuweisungen an die lokalen Variablen ändern die Variablenwerte des aufrufenden Programms nicht. Der einzigste Rückgabewert

ist der Funktionswert, der mit der Return-Anweisung zurück gegeben wird. In derselben Weise wie vorher die Werte können in Zeigervariablen die Adressen von Datenobjekten übergeben werden. Mit der Übergabe von Adressen erhält das Unterprogramm die Möglichkeit, auf die Datenobjekte des aufrufenden Programms lesend und schreibend zuzugreifen. Mit dem Schlüsselwort »const« lässt sich der Zugriff auf »nur lesen« einschränken. Das nachfolgende Unterprogramm zum Kopieren von Zeichenketten bekommt zwei Zeiger übergeben, einen auf den Feldanfang der Datenquelle und einen auf den Feldanfang des Ziels und kopiert, wie im Beispiel zuvor, alle Zeichen einschließlich der abschließenden Null aus dem Zeichenfeld, auf das »src« zeigt, in das Feld, auf das »dest« zeigt:

```
char *strcpy(char *dest, const char *src)
{
 while(*src != 0)
 {
  *dest = *src;
  src++;
  dest++;
 }
 *dest = 0;
 return dest;
}
```

Aufgabe 4.3: Grundfunktionen für die Textverarbeitung

Schreiben Sie die Programmdatei zu dem nachfolgenden Header »string.h« mit folgenden Funktionen zur Textverarbeitung:

```
#ifndef STRING_H
#define STRING_H
char *strcpy(char *dest, const char *src);
char *strcat(char *dest, const char *src);
int strlen(const char *str);
char *append_unsignd(char *dest, int Zahl);
void gets(char *str);
void puts(char *str);
#endif
```

Die Funktion »strcpy« wurde beschrieben. Die Funktion »strcat« soll die Quellzeichenkette an die Zielzeichenkette anhängen. Dazu muss in einer ersten Schleife der Zeiger für die Zielzeichenkette auf die abschließende Null gesetzt werden. Für den anschließenden Kopiervorgang wird im einfachsten Fall »strcpy« mit dem Zielzeiger auf die Abschlussnull aufgerufen. Die Funktion »strlen« soll die Länge der Zeichenkette zurückgeben. Die Funktion »append_unsigned« soll den übergebenen Zahlenwert in eine Zeichenkette umwandeln und zeichenweise an das Ende der Zielzeichenkette kopieren. Die Funktion »gets« soll die von der seriellen Schnittstelle empfangenen Zeichen hintereinander in das Zeichenkettenfeld, auf das »str« zeigt, schreiben. Bei Empfang eines Zeilenumbruchs ist

die Zeichenkette mit einer »0« abzuschließen und zum aufrufenden Programm zurückzukehren. Die Funktion »puts« soll die Zeichenkette, auf die »str« zeigt, zeichenweise über die serielle Schnittstelle versenden. Wenn die abschließende Null erreicht ist, soll ein Zeilenumbruch gesendet und die Funktion beendet werden.

Der Testrahmen sei das Main-Programm mit der üblichen Endlosschleife. In der Endlosschleife sollen nach Eingabeaufforderung zwei Texte eingegeben, hintereinander in eine Zeichenkette geschrieben, der konstante Text »Textlaenge:« und die Textdarstellung der mit »strlen« bestimmten Textlänge angehängt werden:

```
...
char *Feld[100], tmp[80];
while(1)
{
 puts("Eingabetext 1: \n");
 gets(tmp);
 strcpy(Feld, tmp);
 puts("Eingabetext 2: \n");
 gets(tmp);
 strcat(Feld, tmp);
 strcat(Feld, " Textlaenge: ");
 append_unsigned(Feld, strlen(Feld));
 puts(Feld);
}
```

Testbeispiel:

Eingabetext 1:	`Die␣Sonne␣↩`
Eingabetext 2:	`scheint␣hell.↩`
	`Die␣Sonne␣scheint␣hell. Textlaenge: 23`

Kontrollfrage:

- Was passiert bei den implementierten Beispielfunktionen, wenn die mit »strcpy« oder »strcat« erzeugte Zeichenkette länger ist als das Feld, in dem sie gespeichert wird?
- Wie müssten die Funktionen erweitert werden, damit dieser Fehler nicht mehr auftritt?

Aufgabe 4.4: Zeiger und Referenzen

Scheiben Sie ein Main-Programm, das mit folgenden Vereinbarungen und Wertezuweiungen beginnt:

```
int main()
{
```

```
unsigned int x, y, z;
x = 12; y = 5; z = 88;
...
```

und nach jeder Zeicheneingabe folgenden Ausgabetext erzeugt:

```
Variable | Wert | Adresse
-------------------------
    x    |  ?   |    ?
    y    |  ?   |    ?
    z    |  ?   |    ?
```

Statt »?« soll der Wert bzw. die Adresse der Variablen angezeigt werden.

3 Bitverarbeitung

Bei der Ein- und Ausgabe und anderen Hardware-nahen Programmieraufgaben sind häufig einzelne Bits oder Teilbitvektoren zu bearbeiten. Dafür gibt es die bitweisen Logikoperationen und die Verschiebeoperationen:

Negation	UND	ODER	EXOR	Linksverschiebung	Rechtsverschiebung
~a	a & b	a | b	a ^ b	a << n	a >> n

(a, b – typgleiche Konstanten oder Variablen; n – Verschiebung in Bit).

Um z.B. die Leuchtdiode mit der Nummer $n \in \{0, 1, \ldots 7\}$ einzuschalten, bräuchte man eine Variable für den Ausgabewert, zu dem man eine um n Stellen nach links verschobene »1« ODER-verknüpfen und den resultierenden Wert ausgeben würde:

```
char LED;
int n;
...
LED = LED | (1 << n); // Setzen von Bit n
XGpio_WriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0, LED);
```

Zum invertieren wird die ODER-Verknüpfung durch eine EXOR-Verknüpfung und zum ausschalten durch eine UND-Verknüpfung mit dem negierten Wert ersetzt:

```
LED = LED ^ (1 << n); // Invertieren von Bit n
LED = LED & ~(1 << n); // Löschen von Bit n
```

Aufgabe 4.5: Bitverarbeitung

Schreiben Sie ein Programm, dass in einer Endlosschleife auf ein »E« oder »A« von der seriellen Schnittstelle gefolgt von einer Ziffer zwischen »0« und »7« wartet und dann die jeweilige Leuchtdiode ein- bzw. ausschaltet.

Testbeispiel:

Testschritt	Eingabe	Leuchtdioden
0		○○○○○○○○
1	E2	○○○○○●○○
2	E5	○○●○○●○○
3	E4	○○●●○●○○
4	A5	○○○●○●○○
5	E7	●○○●○●○○
6	A2	●○○●○○○○