Microcontroller / C-Programmierung

Selbststudium Semesterwoche 5

Diese Übungen hier lehnen an die Übungen im Leitprogramm (Binärbäume) aus Modul PRG2 (Programmieren 2) an. Entsprechend finden Sie dort zusätzliche hilfreiche Informationen dazu.

1. a) Binärbaum erzeugen
Schreiben Sie ein Programm, welches die Häufigkeit von Wörtern in einem Textfile zählt. Verwenden Sie dabei die Datenstruktur eines Binärbaums. Der Knoten soll dabei einen Zeiger auf ein Wort und die Häufigkeit des Wortes festhalten. Sehen Sie dazu das Beispiel im Buch Seite 134 ff. Es soll eine Funktion programmiert werden, welche ein File ausliest und einen Binärbaum entsprechend aufbaut z.B. mittels Funktion addtree.

1. b) Ausgabe in Preorder
Programmieren Sie eine Funktion, welche in preorder vom Baum die Schlüssel Wort und Häufigkeit ausgibt z.B. mittels printf().

1. c) Suchen im Binärbaum
Schreiben Sie zu dem obigen Binärbaum eine Funktion, welche ein vorgegebenes Wort in dem Binärbaum sucht und als Rückgabewert einen Zeiger auf den Knoten liefert.

2. Aufgabe 6-4 im Buch Seite 138
Schreiben Sie ein Programm, welches die verschiedenen Wörter aus der Eingabe in abnehmender Häufigkeit ihres Auftretens ausgibt. Geben Sie von jedem Wort seine Anzahl aus.

```
// Title: Word Statistics with Pointers
// Author: Thomas Galliker
// Date: 18.10.2009
// Web: http://www.thomasgalliker.ch

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

#define MAXWORDLENGTH 100
#define MAXUNIQUEWORDS 1000

int getword(char *, int);
struct key *binsearch(char *, struct key *, int);
int struct_key_compare_by_count(const void *, const void *);

struct key {
	char *word;
	int count;
};

struct key keytab[MAXUNIQUEWORDS];

void main(void)
{
	char word[MAXWORDLENGTH];
	struct key *p;
	int i = 0;                          // debug value
	int structs_len = 0;       // counts the effective number of elements used in keytab

	/*  fill keytab with unique words */
	while ((i = getword(word, MAXWORDLENGTH)) != EOF)
	{

		if (isalpha(word[0]))
		{
			// try to find the actual word in the input string
			if ((p=binsearch(word, keytab, structs_len)) != NULL)
			{
				// if the actual word was found in the input string
				// we increase the count property of that word
				p->count++;
			}
			else
			{
				// this line is use to get a new address
				char *tmp = (char*) malloc( sizeof(char) );
				strcpy(tmp, word);

				// create a new keytab entry
				keytab[structs_len].word = tmp;
				keytab[structs_len].count = 1;

				// increment
				structs_len++;
			}

		}
	}

	// Sort struct array using qsort (stdlib.h)
    qsort(keytab, structs_len, sizeof(struct key), struct_key_compare_by_count);

	// Print struct array
```

```
	printf("Occurrence\tWord\n");
	printf("--------------------------\n");
	for (p = keytab; p < keytab + structs_len; p++)
	{
		if (p->count > 0)
		{
			printf("%4d\t\t%s\n", p->count, p->word);
		}
	}
	getchar();
}

/* binsearch: find word in tab[0]...tab[n-1] */
struct key *binsearch(char *word, struct key *tab, int n)
{
	int cond;
	struct key *low = &tab[0];
	struct key *high = &tab[n];
	struct key *mid;

	while (low < high)
	{
		// The computation of the middle element can no longer be simply
		// mid = (low+high) / 2
		// because the addition of pointers is illegal.
		// Subtraction is legal, however, so high-low is the number of elements,
		// and thus
		// mid = low + (high-low) / 2
		// sets mid to the element halfway between low and high.
		mid = low + (high-low) / 2;

		if(mid->word != NULL)
		{
			if ((cond = strcmp(tolower(word), tolower(mid->word))) < 0)
				high = mid;
			else if (cond > 0)
				low = mid + 1;
			else
				return mid;
		}
		else
		{
			return NULL;
		}
	}
	return NULL;
}

/* getword: get next word or character from input */
int getword(char *word, int lim)
{
	int c, getch(void);
	void ungetch(int);
	char *w = word;

	while (isspace(c = getchar()))
	;

	if (c != EOF)
	{
		*w++ = c;
	}

	if (!isalpha(c))
	{
		*w = '\0';
		return c;
	}
```

```
	for ( ; --lim > 0; w++)
		if (!isalnum(*w = getchar())) {
			ungetch(*w);
			break;
		}

	*w = '\0';
	return word[0];
}

/* qsort struct comparision function */
int struct_key_compare_by_count(const void *a, const void *b)
{
    struct key *ia = (struct key *)a;
    struct key *ib = (struct key *)b;
       // ib-ia = ascending sort order
       // ia-ib = descending sort order
    return (int)(ib->count - ia->count);
}
```

3. Umwandlung byte[] von/nach float

Sie haben vier Bytes (z.B. via IIC aus einem Flash ausgelesen), welche Sie als float Wert gemäss IEEE 754 interpretieren müssen, resp. umgekehrt. Schreiben Sie Funktionen, welche diese „Umrechnungen“ mit Hilfe einer Union durchführen. Zum Kreieren von Testdaten können Sie zum Beispiel den Rechner von http://www.h-schmidt.net/FloatApplet/IEEE754de.html verwenden.

```
// main.c ///////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>

#include "binToFloat.h"

/*
 * print bytes
 */
void showBinary(unsigned char *bin)
{
	printf("\n2. HEX values:\n     ");
	int i;
	for(i=0; i!=FLOATSIZE; i++)
		printf("0x%02x ",bin[i]);
}

void printLine()
{
	printf("\n---------------------------------------\n");
}

void checkResult(float in_param, float out_param)
{
	printf("\n\n\n\n");
	printLine();
	if(in_param == out_param)
		printf("RESULT: OK");
	else
		printf("RESULT: FAILED");
	printLine();
}
```

```
int main()
{
	unsigned char bin_byte[FLOATSIZE];
	float fIn_param;
	float fout_param;
	while(1) {
		printf("\nEnter number - max length is %d:\n",MAX_IN_LENGTH);
		fIn_param = getFloat();
		if(fIn_param != -1){
			system("cls");
			printLine();
			printf("1. float value entered by user:\n	%f\n",fIn_param);
			makeBin(bin_byte,fIn_param);
			showBinary(bin_byte);

			fout_param = CastToFloat(bin_byte);
			printf("\n\n3. convertet in float:\n	%f",fout_param);
			printLine();
			checkResult(fIn_param,fout_param);
		}
		printf("\n\n\nInstructions:\n	Continue: press 'enter'\n	Exit: press
'ctrl+c'\n");
		getchar();
		system("cls");
	}
	return 0;
}

// binToFloat.h ////////////////////////////////////////////////////////////////
/**
 * Bytes into float-value
 *
 * @authors Zwyssig Markus, Graber Agnes
 * @version 1.0
 */


#ifndef BINTOFLOAT_H_
#define BINTOFLOAT_H_

//defines
#define INPUT_FAILURE	0
#define INPUT_OVERFLOW	1

//makro - converting Float in Binary (without UNION)
#define CNVF(f, n) *((unsigned char*)&(f) + (n))


//Global constants
const static short MAX_IN_LENGTH =	10;
const static short FLOATSIZE = 4;

//Structs and unions
typedef union {
	float f_value;
	unsigned char c_array[4];
}u_data;
```

```
/*************************************************************
 * FILE functions.c
 */

/*
 * Get Flotvalue from user via console
 * @return -1 bei einem Fehler,
 */
float getFloat();

/*
 * Convert float value into a byte array
 */
void makeBin(unsigned char *bin, float f);

/*
 * Convert byte array into float value
 * @param bytearray: unsigned char array
 * @return float value interpreted from bytearray
 */
float CastToFloat(unsigned char* bytearray);

/*************************************************************
 * FILE failures.c
 */

/*
 * show failure message in console
 * @param Failurecode as integer
 */
void showError(int failurecode);


#endif /*BINTOFLOAT_H_*/

// failures.c ////////////////////////////////////////////////////////////////

#include "binToFloat.h"
#include <stdio.h>

void showError(int failurecode)
{
	printf("\n*************\nERROR\n*************\n");
	if(failurecode == INPUT_FAILURE)
		printf("Falsche Eingabe\n");
	else if(failurecode == INPUT_OVERFLOW)
		printf("Input Overflow\n");
	printf("\n\n");
}
```

```
// functions.c ////////////////////////////////////////////////////////////////

#include "binToFloat.h"
#include <stdio.h>
#include <stdlib.h>


//-1 wird als fehler intepretiert
float getFloat()
{
	int i;
	float f;
	char *sInputString = (char *) malloc(sizeof(sInputString)*MAX_IN_LENGTH);
	gets(sInputString);
	for(i=0; sInputString[i]!=0; i++)
		if (i >= MAX_IN_LENGTH){
			showError(INPUT_OVERFLOW);
			return -1;
		}
	f = atof(sInputString);
	free(sInputString);
	if(f == 0){
		showError(INPUT_FAILURE);
		return -1;
	}
	return f;
}

void makeBin(unsigned char *bin, float f)
{
	int i;
	for(i = 0; i < FLOATSIZE; i++)
		bin[i] = CNVF(f, i);
}

float CastToFloat(unsigned char* bytearray)
{
	u_data data;
	int i=0;
	for(i=0; i!=FLOATSIZE; i++) {
		data.c_array[i] = bytearray[i];
	}
	return data.f_value;
}
```

4. Matrixen (pointer-to-pointer) *

Schreiben Sie die Funktion createMatrix(…), welche eine Matrix erzeugt und die Matrix mit einem Werte initialisiert. Der untenstehende Aufruf erzeugt eine 3x4 Matrix mit dem Inhalt 111 111 111 111 111 111 111 111 111 111 111 111 int **m; m = createMatrix(3, 4, 111); 1. Parameter: Anzahl Zeilen 2. Parameter: Anzahl Spalten 3. Parameter: Initialisierungswerte für die Matrix

```
#include <stdio.h>


int **createMatrix(int rows, int cols, int value)
{
	int x, y;
	int **matrix = (int**)malloc(sizeof(int) * rows);
	if(matrix!=0)
	{
		for(x = 0; x<rows; x++)
		{
			*(matrix+x) = (int*)malloc(sizeof(int) * cols);
			for(y = 0; y<cols; y++)
			{
				//matrix[x][y] = value;
				*(*(matrix+x)+y) = value;
			}
		}
		return matrix;
	}
}

int printMatrix(int ***matrix, int rows, int cols)
{
	int x, y;
	if(matrix!=0)
	{
		for(x = 0; x<rows; x++)
		{
			for(y = 0; y<cols; y++)
			{
				printf("%d ", *(*(*matrix+x)+y));
			}
			printf("\n");
		}
	}
}

void main(void)
{
	int **m = createMatrix(6, 3, 1);
	printMatrix(&m, 6, 3);

	getchar();
	getchar();
}
```