Übung 6.1 – Projekt Adressverwaltung

In dieser Übung vertiefen Sie noch einmal viele Elemente der Programmiersprache C, so zum Beispiel Strukturen und Zeiger auf Strukturen, und Sie können viele Funktionen zu Eingabe, Ausgabe und Dateiverwaltung anwenden.

Auftrag:

Schreiben Sie ein Programm zur Adressverwaltung. Über die Konsole können Sie neue Adressen eingeben, die Adressliste ausgeben und nach verschiedenen Kriterien sortieren. Erweitern Sie das Ihre Adressverwaltung, sodass die Adressliste in eine Datei abgespeichert und wieder gelesen werden kann.

main.c

```
/**============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
--------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
              Schreiben Sie ein Programm zur Adressverwaltung. Über die Konsole können Sie neue
              Adressen eingeben, die Adressliste ausgeben und nach verschiedenen Kriterien
              sortieren. Erweitern Sie das Ihre Adressverwaltung, sodass die Adressliste in eine
              Datei abgespeichert und wieder gelesen werden kann.
============================================================================================*/
#include <stdlib.h>
#include "address.h"
#include "addressConsoleIO.h"

/*
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv) {
    err_t errCode = 0; // errorcode handler

    // init AddressList
    errCode = initialize();
    if (errCode > 0) {
        //printf("Init: Successfully...\n");
        showMenu(TRUE);
    } else {
        //printf("Error: Init AddrList: (Err: %d)\n", errCode);
        return (EXIT_FAILURE);
    }

    return (EXIT_SUCCESS);
}
```

address.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                    Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                     http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : address.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
============================================================================================================*/
#ifndef ADDRESS_H
#define ADDRESS_H

#define PATH_SIZE 255 // File Path Buffer Size
#define TRUE      1   // Boolean True
#define FALSE     0   // Boolean False

/**
 * Custom Error Return Code
 */
typedef enum errorCode {
    SUCCESS           = 1,   // OK
    UNKNOW            = 0,   // undef status
    ERROR             = -1,  // general error
    NOMEMORY          = -2,  // alloc, realloc failed
    FILECREATEERROR   = -11, // fopen failed
    FILECLOSEERROR    = -12, // fclose failed
    FILENOTFOUNDERROR = -13, // file not found
    FILEREADERROR     = -14  // file read failed
} err_t;

/**
 * Defines address struct
 */
#define MAXSTRLEN 45 // Max String Len
typedef struct address_ {
    char firstname[MAXSTRLEN+1];
    char lastname[MAXSTRLEN+1];
    char street[MAXSTRLEN+1];
    unsigned int zip;
    char city[MAXSTRLEN+1];
    unsigned int index;
} address_t, address_p;

/**
 * Defines address List
 */
typedef struct addressList_ {
    address_t* addr;
    int size;
} addressList_t, addressList_p;

/**
 * Init AddressList (Allocate Memory)
 * @return Success / Error Status Code
 */
err_t initialize(void);

/**
 * List the current Address List (with CallBack Function for Output)
 * @param showAddrEntryFN CallBack Function for Output
 */
void listAddress(void (*showAddrEntryFN)(int, address_t*));

/**
 * Add a new Address to the List (Values)
 * @param lastname Lastname of the new entry
 * @param firstname Firstname of the new entry
 * @param street Street of the new entry
 * @param zip Zip of the new entry
 * @param city City of the new entry
 * @return Success / Error Status Code
 */
err_t addAddressValues(char* lastname, char* firstname, char* street, int zip, char* city);

/**
 * Add a new Address to the list (Address Object Pointer)
 * @param newAddress Pointer to the new Address
 * @return Success / Error Status Code
```

```
 */
err_t addAddress(address_p newAddress);

/**
 * Clear Address List in the Memory (free & re-malloc)
 * @return Success / Error Status Code
 */
err_t clearAddress(void);

/**
 * Save the current AddressList to a File
 * @param path Path to File for save
 * @param num Number or Addresses Loaded
 * @return Success / Error Status Code
 */
err_t saveAddrFile(char* path, int* num);

/**
 * Load Addresses from a File and them to the current AddressList
 * @param path Path to File for reading
 * @param num Number or Addresses Loaded
 * @return Success / Error Status Code
 */
err_t readAddrFile(char* path, int* num);

/**
 * Defines SortMode
 */
typedef enum { NAME=1, STREET, ZIP, CITY} sortmode;  // SortMode

/**
 * Sort Address by Sortmode with a modified (stable!) QuickSort
 * @param mode Sortmode, i.e. NAME, STREET, ZIP or CITY
 * @return Success / Error Status Code
 */
err_t sortAddress(sortmode mode);

/**
 * Compare Function pointer
 */
typedef int (*compfn)(const void *a, const void *b);

/**
 * Comparator by Name
 * @param a Address 1
 * @param b Address 2
 * @return Compare Result, >0 if Addr2 is bigger than Addr1
 */
int compareByName(const void *a, const void *b);

/**
 * Comparator by Street
 * @param a Address 1
 * @param b Address 2
 * @return Compare Result, >0 if Addr2 is bigger than Addr1
 */
int compareByStreet(const void *a, const void *b);

/**
 * Comparator by Zip
 * @param a Address 1
 * @param b Address 2
 * @return Compare Result, >0 if Addr2 is bigger than Addr1
 */
int compareByZip(const void *a, const void *b);

/**
 * Comparator by City
 * @param a Address 1
 * @param b Address 2
 * @return Compare Result, >0 if Addr2 is bigger than Addr1
 */
int compareByCity(const void *a, const void *b);

#endif  /* ADDRESS_H */
```

address.c

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : address.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
==============================================================================================*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "address.h"
#include "addressFileIO.h"

// Global VAR: Addresslist
addressList_t* addrList = NULL;

err_t initialize(void)
{
    //create a new List
    addrList = (addressList_t*) malloc(sizeof(addressList_t));
    if (addrList) { // allocate Memory successfully?
        addrList->addr = NULL;
        addrList->size = 0;
        return SUCCESS;
    } else {
        return NOMEMORY;
    };
}

void listAddress(void (*showAddrEntryFN)(int, address_t*))
{
    int i; // tmp counter

    for (i=0; i<addrList->size; i++) {
        showAddrEntryFN(i+1, &(addrList->addr[i])); // Use CallBack Function for Output
    }
}

err_t addAddressValues(char* lastname, char* firstname, char* street, int zip, char* city)
{
    address_p newAddr; // Address Object to fill and add

    strcpy(newAddr.firstname, firstname);
    strcpy(newAddr.lastname, lastname);
    strcpy(newAddr.street, street);
    newAddr.zip = zip;
    strcpy(newAddr.city, city);
    return addAddress(newAddr);
}

err_t addAddress(address_p newAddress)
{
    int count; // Address Counter

    count = addrList->size;
    if (count == 0) {
        // alloc new memory
        addrList->addr = (address_t*) malloc(sizeof(address_t));
    } else {
        // realloc memory
        addrList->addr = (address_t*) realloc(addrList->addr, sizeof(address_t) * (count + 1));
    }
    if (addrList->addr) {
        // Add new Node
        strcpy(addrList->addr[count].firstname, newAddress.firstname);
        strcpy(addrList->addr[count].lastname, newAddress.lastname);
        strcpy(addrList->addr[count].street, newAddress.street);
        addrList->addr[count].zip = newAddress.zip;
        strcpy(addrList->addr[count].city, newAddress.city);
        addrList->addr[count].index = count;
        addrList->size = ++count;
        return SUCCESS;
    } else {
        return NOMEMORY;
    }
```

```
}

err_t clearAddress(void)
{
    // release memory
    free(addrList);
    // alloc new memory
    return initialize();
}

err_t saveAddrFile(char* path, int* num)
{
    return writeFile(path, addrList, formatCSVTab, num);
}

err_t readAddrFile(char* path, int* num)
{
    return readFile(path, addrList, lineParser, num);
}

err_t sortAddress(sortmode mode)
{
    int i; // temp counter, used for stable sort

    // qsort(Beginning address of array, Number of elements, Size of each element, Pointer to compare
function
    switch (mode) {
        case NAME:
            qsort(addrList->addr,addrList->size, sizeof(address_t), (compfn)compareByName);
            break;
        case STREET:
            qsort(addrList->addr,addrList->size, sizeof(address_t), (compfn)compareByStreet);
            break;
        case ZIP:
            qsort(addrList->addr,addrList->size, sizeof(address_t), (compfn)compareByZip);
            break;
        case CITY:
            qsort(addrList->addr,addrList->size, sizeof(address_t), (compfn)compareByCity);
            break;
    }
    // update index - used for stable sort
    for (i=0; i<addrList->size; i++) {
        addrList->addr[i].index = i;
    }
    return SUCCESS;
}

int compareByName(const void *a, const void *b)
{
    int res;                            // errorcode handler
    char name1[2*MAXSTRLEN + 1];        // Buffer for address 1 Name
    char name2[2*MAXSTRLEN + 1];        // Buffer for address 2 Name
    address_t *addr1 = (address_t*) a; // address 1, cast param
    address_t *addr2 = (address_t*) b; // address 2, cast param

    // Copy Lastname and Firstname into one String
    strcpy(name1, addr1->lastname);
    strcat(name1, addr1->firstname);
    strcpy(name2, addr2->lastname);
    strcat(name2, addr2->firstname);

    // Compare String Value 'Lastname Firstname'
    res = strcasecmp(name1, name2); // Compare String (ignore case)
    if (res == 0) { // stable sort
        res = addr1->index - addr2->index;
    }
    return res;
}

int compareByStreet(const void *a, const void *b)
{
    int res;                            // errorcode handler
    address_t *addr1 = (address_t*) a; // address 1, cast param
    address_t *addr2 = (address_t*) b; // address 2, cast param

    // Compare String Value 'Street'
    res = strcasecmp(addr1->street, addr2->street); // ignore case
    if (res == 0) { // stable sort
        res = addr1->index - addr2->index;
    }
```

```
    return res;
}

int compareByZip(const void *a, const void *b)
{
    int res;                            // errorcode handler
    address_t *addr1 = (address_t*) a; // address 1, cast param
    address_t *addr2 = (address_t*) b; // address 2, cast param

    // Compare Int value 'Zip'
    res = addr1->zip - addr2->zip;
    if (res == 0) { // stable sort
        res = addr1->index - addr2->index;
    }
    return res;
}

int compareByCity(const void *a, const void *b)
{
    int res;                            // errorcode handler
    address_t *addr1 = (address_t*) a; // address 1, cast param
    address_t *addr2 = (address_t*) b; // address 2, cast param

    // Compare String Value 'City'
    res = strcasecmp(addr1->city, addr2->city);
    if (res == 0) { // stable sort
        res = addr1->index - addr2->index;
    }
    return res;
}
```

addressConsoleIO.h

```
/**=============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                          http://hslu.ximit.ch
-------------------------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : addressConsoleIO.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
              Adressen aus der Konsole einlesen, bestehende ausgeben
=============================================================================================================*/
#ifndef ADDRESSCONSOLEIO_H
#define ADDRESSCONSOLEIO_H

/**
 * Main Menu
 * @param showMainMenu Use Large or Small Menu
 */
void showMenu(int showMainMenu);

/**
 * Show Main Menu Desc: Large
 */
void showMainMenuLarge(void);

/**
 * Show Main Menu Desc: Small
 */
void showMainMenuSmall(void);

/**
 * GUI Add a new Address
 */
void addAddressGUI(void);

/**
 * GUI Save the current AddressList to a File
 */
void saveAddrFileGUI(void);

/**
 * GUI Load Addresses from a File and them to the current AddressList
 */
void readAddrFileGUI(void);

/**
 * GUI Sort Address by Sortmode with a modified (stable!) QuickSort
 * @param mode Sortmode, i.e. NAME, STREET, ZIP or CITY
 */
void sortAddressGUI(sortmode mode);

/**
 * GUI Clear Address List in the Memory (free & re-malloc)
 */
void clearAddressGUI(void);

/**
 * GUI Show one Address (CallBack Function)
 * @param count Index #Number
 * @param addr Addres to display
 */
void displayAddr(int count, address_t* addr);

/**
 * Show the current Address List
 */
void listAddressGUI(void);

#endif  /* ADDRESSCONSOLEIO_H */
```

addressConsoleIO.c

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : addressConsoleIO.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
              Adressen aus der Konsole einlesen, bestehende ausgeben
=============================================================================================*/
#include <stdio.h>
#include <string.h>
#include "address.h"
#include "addressConsoleIO.h"

void showMenu(int displargeMenu)
{
    char menu = ' '; // buffer for key, user menu choice
    int listAfterSort = TRUE;

    do // Menu (until Q pressed)
    {
        if (displargeMenu) {
            // show Main Menu Desc (large)
            showMainMenuLarge();
            displargeMenu = FALSE;
        } else {
            // show Main Menu Desc (small)
            showMainMenuSmall();
        }
        printf(" » Please choose: ");

        while (!isalnum(menu = getchar())); // Solange einlesen bis Zahl oder Buchstabe
        menu = toupper(menu);

        switch(menu){
        case 'N': // Insert a new address
            addAddressGUI();
            break;
        case 'L': // List stored addresses
            listAddressGUI();
            break;
        case 'C': // Clear stored addresses
            clearAddressGUI();
            break;
        case 'R': // Read addresses from file
            readAddrFileGUI();
            break;
        case 'S': // Save addresses to file
            saveAddrFileGUI();
            break;
        case 'M': // Show Main Menu
            displargeMenu = TRUE;
            break;
        case '1': // Sort addresses by name & List them
            sortAddressGUI(NAME);
            break;
        case '2': // Sort addresses by street & List them
            sortAddressGUI(STREET);
            break;
        case '3': // Sort addresses by zip & List them
            sortAddressGUI(ZIP);
            break;
        case '4': // Sort addresses by city & List them
            sortAddressGUI(CITY);
            break;
        }
    } while(menu != 'Q');
}
```

```
void showMainMenuLarge(void)
{
        printf("\n");
        printf("╔═══════════════════════════════════════════════════════════════════════════════╗\n");
        printf("║                                                                               ║\n");
        printf("║       _       _   _                  __  __                                   ╫\n");
        printf("║      /_\\  __| |__| |_ _ ___ ______  |  \\/  |__ _ _ _  __ _ __ _ ___ _ _      ╫\n");
        printf("║     / _ \\/ _` / _` | '_/ -_|_-<_-< | |\\/| / _` | ' \\/ _` / _` / -_) '_|     ╫\n");
        printf("║    /_/ \\_\\__,_\\__,_|_| \\___/__/__/ |_|  |_\\__,_|_||_\\__,_\\__, \\___|_|       ╫\n");
        printf("║                                                            |___/              ║\n");
        printf("╠═══════════════════════════════════════════════════════════════════════════════╣\n");
        printf("╟   Select an option                                                            ╢\n");
        printf("╟     N : Insert a new address         1 : Sort addresses by name              ╢\n");
        printf("╟     L : List stored addresses        2 : Sort addresses by street            ╢\n");
        printf("╟     C : Clear stored addresses       3 : Sort addresses by zip               ╢\n");
        printf("╟                                      4 : Sort addresses by city              ╢\n");
        printf("╟     R : Read addresses from file                                             ╢\n");
        printf("╟     S : Save addresses to file       Q : Quit           © 2012 Felix Rohrer  ╢\n");
        printf("╚═══════════════════════════════════════════════════════════════════════════════╝\n");
}

void showMainMenuSmall(void)
{
        printf("\n");
        printf("╔═══════════════════════════════════════════════════════════════════════════════╗\n");
        printf("╟ N: New ┃ L: List ┃ C: Clear ┃ R: ReadFile ┃ S: SaveFile ┃ M: Menu ┃ Q: Quit ╢\n");
        printf("╟═══════════════════════════════════════════════════════════════════════════════╢\n");
        printf("╟ 1: sort by name ┃ 2: sort by street ┃ 3: sort by zip ┃ 4: sort by city        ╢\n");
        printf("╚═══════════════════════════════════════════════════════════════════════════════╝\n");
}

void addAddressGUI(void)
{
    int res;                    // errorcode handler
    char firstname[MAXSTRLEN+1]; // Buffer for Firstname
    char lastname[MAXSTRLEN+1];  // Buffer for Lastname
    char street[MAXSTRLEN+1-5];  // Buffer for Street
    char streetNbr[5+1];         // Buffer for Street Number
    unsigned int zip;            // Buffer for Zip
    char city[MAXSTRLEN+1];      // Buffer for City
    size_t length;               // Lenght of Street, combine Street + StreetNbr

    printf("\n\n┌──────────────────────────────────────────────────────────────────────────────┐\n");
    printf("│ Add new address                                                              │\n");
    printf("├──────────────────────────────────────────────────────────────────────────────┘\n");
    printf("│ Firstname:  ");
    scanf("%s", firstname);
    printf("│ Lastname:   ");
    scanf("%s", lastname);
    printf("│ Street:     ");
    scanf("%s", street);
    printf("│ Street Nbr: ");
    scanf("%s", streetNbr);
    printf("│ Zip:        ");
    scanf("%i", &zip);
    printf("│ City:       ");
    scanf("%s", city);
    printf("├──────────────────────────────────────────────────────────────────────────────\n");

    // combine street & streetnumber
    length = strlen(street);
    street[length] = ' ';
    street[length+1] = '\0';
    strcat(street, streetNbr);
    // add new addr.
    res = addAddressValues(lastname, firstname, street, zip, city);
    if (res != SUCCESS) {
        printf("│ ERROR: Could not add new address! (Err: %i)\n", res);
    } else {
        printf("│ New address added successfully.\n");
    }
    printf("└──────────────────────────────────────────────────────────────────────────────\n");
}

void saveAddrFileGUI(void)
{
    err_t res;            // errorcode handler
    char path[PATH_SIZE]; // filepath, filename buffer
    int count = 0;        // counter of addresses

    printf("\n\n┌──────────────────────────────────────────────────────────────────────────────\n");
```

```
    printf("| Save to file                                                                 |\n");
    printf("|------------------------------------------------------------------------------|\n");
    printf("| Please enter filename: ");
    scanf("%s", path);
    res = saveAddrFile(path, &count); // Save AddressList to a File
    if (res != SUCCESS) {
        printf("| ERROR: Could not write file! (Err: %i)\n", res);
    } else {
        printf("| Save done. %i addresses saved into file '%s'\n", count, path);
    }
    printf("└------------------------------------------------------------------------------┘\n");
}

void readAddrFileGUI(void)
{
    err_t res;            // errorcode handler
    char path[PATH_SIZE]; // filepath, filename buffer
    int count = 0;        // counter of addresses

    printf("\n\n┌------------------------------------------------------------------------------┐\n");
    printf("| Load from file                                                               |\n");
    printf("|------------------------------------------------------------------------------|\n");
    printf("| Please enter filename: ");
    scanf("%s", path);
    res = readAddrFile(path, &count); // Read Addresses from a File and add them to the current AddressList
    if (res != SUCCESS) {
        printf("| ERROR: Could not read file '%s'! (Err: %i)\n", path, res);
    } else {
        printf("| Load done. %i addresses loaded from file '%s'.\n", count, path);
    }
    printf("└------------------------------------------------------------------------------┘\n");
}

void sortAddressGUI(sortmode mode)
{
    err_t res; // errorcode handler

    res = sortAddress(mode); // SortAddress by SortMode
    if (res == SUCCESS) {
        printf(" » Address List successfully sorted...\n");
    } else {
        printf(" » ERROR: Address List could not be sorted! (Err: %i)\n", res);
    }
}

void clearAddressGUI(void)
{
    err_t res; // errorcode handler

    res = clearAddress(); // free & re-malloc memory
    if (res == SUCCESS) {
        printf(" » Address List cleared...\n");
    } else {
        printf(" » ERROR: Address List could not be cleared! (Err: %i)\n", res);
    }
}

void displayAddr(int count, address_t* addr)
{
    char nameBuff[2*MAXSTRLEN+1]; // Buffer for Lastname + Firstname

    sprintf(nameBuff, "%s %s", addr->lastname, addr->firstname);
    printf("| %4i %-25.24s%-25.24s%-5d%.16s\n",
            count,
            nameBuff,
            addr->street,
            addr->zip,
            addr->city);
}

void listAddressGUI(void)
{
    printf("\n\n┌------------------------------------------------------------------------------┐\n");
    printf("| Address List                                                                 |\n");
    printf("|------------------------------------------------------------------------------|\n");
    printf("| %4s %-25s%-25s%-5s%s\n", "#", "Name", "Street", "Zip", "City");
    listAddress(displayAddr); // ListAddress with CallBack Function displayAddr -> Console Stdout
    printf("└------------------------------------------------------------------------------┘\n");
}
```

addressFileIO.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                       Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                        http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : addressFileIO.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
              Adressen in eine Datei (csv) speichern oder aus einer Datei importieren.
============================================================================================================*/
#ifndef ADDRESSFILEIO_H
#define ADDRESSFILEIO_H

/**
 * Parse the line to get the Address Details
 * @param line Addres Line from CSV Export
 * @return Success / Error Status Code
 */
err_t lineParser(char* line);

/**
 * Export AddressList to a CSV_File
 * @param path Filepath, Filename for the csv-File
 * @param addrList Pointer to the current AddressList
 * @param formatCSVfn CallBack-Function to format the Output
 * @param num Return value how many Addresses are exporet
 * @return Success / Error Status Code
 */
err_t writeFile(char* path, addressList_t* addrList, err_t (*formatCSVfn)(address_t*, char*), int* num);

/**
 * Import AddressList from a CSV-File
 * @param path Filepath, Filename for the csv-File
 * @param addrList Pointer to the current AddressList
 * @param lineParserFn CallBack-Function for the LineParser (Format)
 * @param num Returnh value how many Addresses are imported
 * @return Success / Error Status Code
 */
err_t readFile(char* path, addressList_t* addrList, err_t (*lineParserFn)(char*), int* num);

/**
 * CallBack-Function to Format the Export: CSV by Tab
 * @param addr Pointer to one Address
 * @param line Formated Output String
 * @return Success / Error Status Code
 */
err_t formatCSVTab(address_t* addr, char* line);

/**
 * CallBack-Function to Format the Export: CSV by Semicolon
 * @param addr Pointer to one Address
 * @param line Formated Output String
 * @return Success / Error Status Code
 */
err_t formatCSVSemicolon(address_t* addr, char* line);

#endif  /* ADDRESSFILEIO_H */
```

addressFileIO.c

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                       Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                        http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Adressverwaltung
Name        : addressFileIO.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-19v1.0
Description : Adressverwaltung
              Adressen in eine Datei (csv) speichern oder aus einer Datei importieren.
============================================================================================================*/
#include <stdio.h>
#include <string.h>
#include "address.h"
#include "addressFileIO.h"

#define BUFFER_SIZE 2048 // File Line Buffer Size

err_t lineParser(char* txtLine)
{
    char lastname[MAXSTRLEN+1];  // buffer for lastname
    char firstname[MAXSTRLEN+1]; // buffer for firstname
    char street[MAXSTRLEN+1];    // buffer for street
    int zip;                     // buffer for zip
    char city[MAXSTRLEN+1];      // buffer for city
    char* txtPart;               // buffer for text token

    // init, set default to empty
    strcpy(firstname, "\0");
    strcpy(lastname, "\0");
    strcpy(street, "\0");
    zip = -1;
    strcpy(city, "\0");

    // Reads all address tokens out of txtLine
    // lastname
    txtPart = strtok(txtLine, "\t"); // only first read needs the txtLine
    if (txtPart != NULL && (strlen(txtPart)<MAXSTRLEN)) {
        strcpy(lastname, txtPart);
    } else { return FILEREADERROR; }

    // firstname
    txtPart = strtok(NULL, "\t");
    if (txtPart != NULL && (strlen(txtPart)<MAXSTRLEN)) {
        strcpy(firstname, txtPart);
    } else { return FILEREADERROR; }

    // street
    txtPart = strtok(NULL, "\t");
    if (txtPart != NULL && (strlen(txtPart)<MAXSTRLEN)) {
        strcpy(street, txtPart);
    } else { return FILEREADERROR; }

    // zip
    txtPart = strtok(NULL, "\t");
    if (txtPart != NULL && (strlen(txtPart)<MAXSTRLEN)) {
        zip = atoi(txtPart);
    } else { return FILEREADERROR; }

    // city
    txtPart = strtok(NULL, "\r"); // \r to remove trailing cr / newline
    if (txtPart != NULL && (strlen(txtPart)<MAXSTRLEN)) {
        strcpy(city, txtPart);
    } else { return FILEREADERROR; }

    // add new Address
    return addAddressValues(lastname, firstname, street, zip, city);
}

err_t writeFile(char* path, addressList_t* addrList, err_t (*formatCSVfn)(address_t*, char*), int* num)
{
    int res;                // errorcode handler
    int count;              // counter of addresses
    int i;                  // tmp counter
    FILE *file = NULL;      // file handler
    char line[BUFFER_SIZE]; // buffer for one line

    // create new file (overwrite existing one)
    file = fopen(path, "w+");
```

```
    if (file == NULL) {
        return FILECREATEERROR;
    }
    // write file
    count = addrList->size;
    *num = count; // save number of addresses for output
    for (i=0; i<count; i++) {
        res = (*formatCSVfn)(&(addrList->addr[i]), line); // Use CallBack-Func. to Format the Export
        if (res != SUCCESS || fputs(line, file) == EOF) {
            free(line);
            fclose(file);
            return res;
        }
    }
    // close the file, define return status
    if (fclose(file) != 0) {
        res = FILECLOSEERROR;
    } else {
        res = SUCCESS;
    }
    return res;
}

err_t readFile(char* path, addressList_t* addrList, err_t (*lineParserFn)(char*), int* num)
{
    err_t res;              // errorcode handler
    FILE *file = NULL;      // file handler
    char line[BUFFER_SIZE]; // buffer file reading
    int count = 0;          // Counter for addresses

    // Open the file (read-only)
    file = fopen(path, "r");
    if (file == NULL) {
        return FILENOTFOUNDERROR;
    }

    // Reads until no more lines are available in the csv
    while(fgets(line, sizeof(line), file) != NULL) {
        count++;
        // try to parse and process the file (Use CallBack-Func.)
        res = (*lineParserFn)(line);
        // in case of error, close the file and abort with error code
        if (res != SUCCESS) {
            fclose(file);
            file = NULL;
            return res;
        }
    }

    // close the file
    if (fclose(file) != 0) {
        res = FILECLOSEERROR;
    } else {
        res = SUCCESS;
    }
    *num = count; // save back counter value
    return res;
}

err_t formatCSVTab(address_t* addr, char* line){
    // format string for csv export
    sprintf(line, "%s\t%s\t%s\t%i\t%s\r\n",
            addr->lastname,
            addr->firstname,
            addr->street,
            addr->zip,
            addr->city);
    return SUCCESS;
}

err_t formatCSVSemicolon(address_t* addr, char* line){
    // format string for csv export
    sprintf(line, "%s;%s;%s;%i;%s\r\n",
            addr->lastname,
            addr->firstname,
            addr->street,
            addr->zip,
            addr->city);
    return SUCCESS;
}
```