Personenverwaltung

Zusammenfassung der Problemstellung

Das Programm soll eine einfache Personenverwaltung simulieren. Dem Benutzer stehen jederzeit 5 Programmoptionen zur Verfügung:

- Neue Person hinzufügen
- Bestehende Person löschen
- Liste mit allen erfassten Personen anzeigen
- Liste löschen
- Programm beenden

Das Programm läuft also solange, bis der Benutzer das Programm beendet.

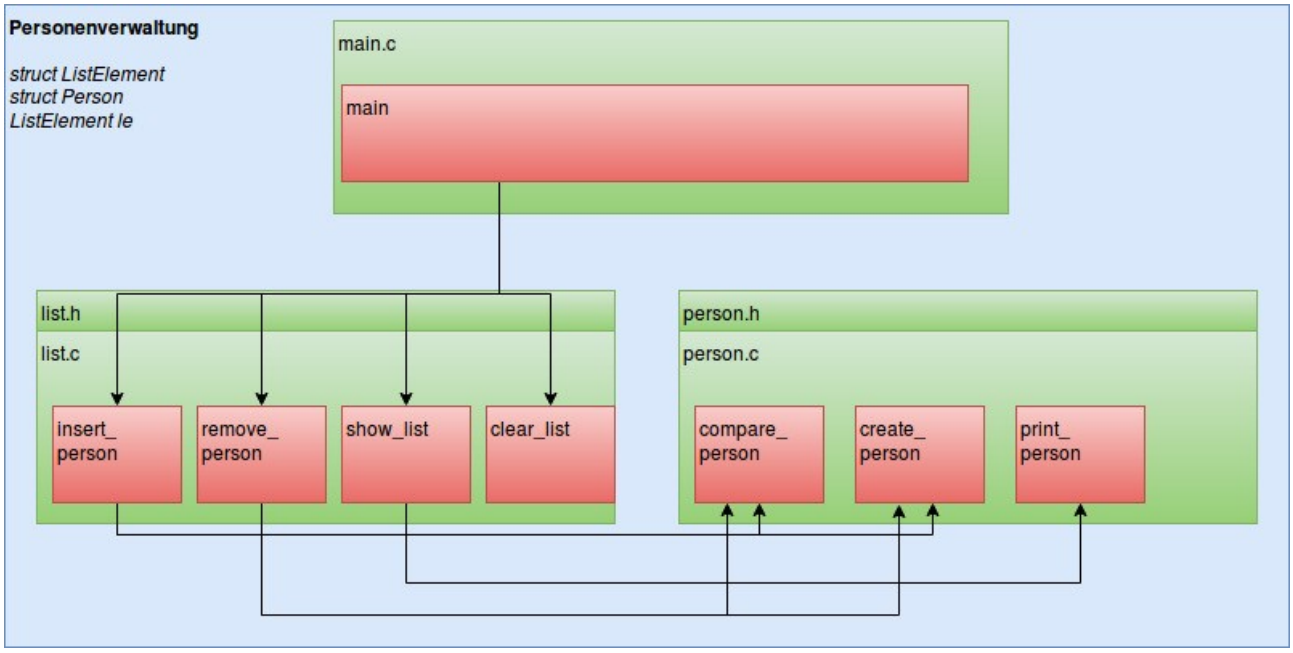
Lösungsvarianten

In einer while-Schleife werden mit einem switch-Statement die Inputs des Benutzers entgegengenommen. Die Schleife läuft solange, bis der Benutzer das Programm mittels Eingabe von ‚E' (End) beendet. Mit dieser Lösungsvariante ist es einfach möglich, neue Befehle hinzuzufügen.

Für das Speichern der Personendaten werden 2 structs definiert: ListElement und Person, wobei ListElement jeweils die Person und den Verweis auf das nächste ListElement speichert. Die Daten werden also in einer einfach verketteten Liste gespeichert. Das letzte Element zeigt wieder auf das Startelement, so dass ein Ring entsteht. Diese Listen-Variante wurde in der Aufgabestellung so vorgegeben. Alternativ hätten auch andere Listen (z.B. ArrayList) verwendet werden können.

Mit der gewählten Lösung ist es möglich, beliebig viele Personendaten zu speichern, solange genügend Speicherplatz zur Verfügung steht. Der Speicher wird dabei dynamisch mittels malloc alloziert und beim Löschen eines Datensatzes mittels free auch wieder freigegeben.

Modulübersicht



Das Programm ist von seiner Funktionalität her zu gross, um alle benötigten Bausteine in eine Quellcode-Datei zu packen. Wir entscheiden uns für die in der Aufgabenstellung vorgeschlagene Variante der Aufteilung in 3 Quellcode- und 2 Headerdateien:

- main.c
  - ruft auf insertPerson, removePerson, showList, clearList
- list.c
  - insertPerson(ruft auf createPerson, comparePerson), removePerson(ruft auf createPerson, comparePerson), showList (ruft auf printPerson), clearList
- list.h
- person.c
  - comparePerson, createPerson, printPerson
- person.h

Globale Variablen: ListElement le
Structs: ListElement, Person

Programmablauf

Main > Benutzer hat hier die Wahl zwischen den 5 Programmfunktionen > Durch Einlesen des eingegebenen Zeichens wird die entsprechende Funktion (alle Hauptfunktionen befinden sich in list.c) aufgerufen > Die Funktion wird abgearbeitet und ruft allenfalls Hilfsfunktionen aus person.c auf > Das Programm kehrt zu main zurück > dies wird so fortgesetzt bis der Benutzer das Programm mit der entsprechenden Eingabe terminiert

Tests

Es wurden 3 Tests geschrieben. Die 2 ersten Tests simulieren jeweils eine Abfolge von User-Eingaben: Einfügen von Personendaten, Löschen von Personendaten und Anzeigen der Liste. Beim zweiten Test wird vor der Anzeige die Liste gelöscht. So können alle Programmfunktionen getestet werden.

Beim dritten Test wird die Hilfsmethode comparePerson getestet. Hierbei handelt es sich um einen klassischen UnitTest.

Alle 3 Tests liefen erfolgreich durch.

Erkenntnisse

Das Programm läuft fehlerfrei. Weitere Implementationen wie z.B. die Erweiterung der erfassten Personendaten (z.B. Adresse, Telefonnummer) oder das Editieren bereits erfasster Daten wären ohne grossen Aufwand möglich.

Anhang A: Quelltext Personenverwaltung

Main.c

```
/* ---------------------------------------------------------------------------
 * --  _____       _______  _____                                        -
 * -- |_   _|     |  _____|/ ____|                                       -
 * --   | |  _ __ | |__  | (___    Institute of Embedded Systems          -
 * --   | | | '_ \|  __|  \___ \   Zuercher Hochschule Winterthur        -
 * --  _| |_| | | | |____ ____) |  (University of Applied Sciences)      -
 * -- |_____|_| |_|_______|_____/   8401 Winterthur, Switzerland         -
 * ---------------------------------------------------------------------------
 */
/**
 * @file
 * @brief program to store person data in a list, providing the options to
 *        insert and remove persons and to show and clear the list.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <list.h>
#include <person.h>

ListElement le;

int main(void)
```

```
{
   le.next = &le;
   char inputChar = ' ';
   while (inputChar != 'E') {
      (void)printf("I(nsert), R(emove), S(how), C(lear), E(nd):\n");
      (void)scanf(" %c", &inputChar);
      switch(inputChar) {
         case 'I': (void)insert_person();
         break;
         case 'R': (void)remove_person();
         break;
         case 'S': (void)show_list();
         break;
         case 'C': (void)clear_list();
         break;
      }
   }
}
```

List.h

```
#ifndef LIST_H
#define LIST_H

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

typedef struct LE ListElement;

struct LE {
   Person content;
   ListElement *next;
};

ListElement le;

void insert_person(void);
void remove_person(void);
void show_list(void);
void clear_list(void);

#endif
```

List.c

```
#include <list.h>

/*
```

```
** function to insert a new person to the list.
**
*/
void insert_person(void) {
   // create new person
   Person newPerson = create_person();
   // go to correct position in the list
   ListElement *pointer = &le;
   while (pointer->next != &le &&
         compare_person(newPerson, pointer->next->content) > 0 ) {
      pointer=pointer->next;
   }
   // create new list element and point it to the new person
   ListElement *temp = pointer->next;
   pointer->next = malloc(sizeof(ListElement));
   pointer=pointer->next;
   pointer->content = newPerson;
   pointer->next = temp;
   // print out info on newly added person
   (void)printf("Person added successfully: %s %s, %d\n",
            newPerson.name, newPerson.firstname, newPerson.age);
}

/*
** function to remove a person from the list.
**
*/
void remove_person(void) {
   // create person object with the attributes of the person that
   // shall be removed.
   Person to_remove = create_person();
   // go through the list and look for a match.
   ListElement *pointer = &le;
   int success = 0;
    while (pointer->next != &le && success != 1) {

      if (compare_person(to_remove, pointer->next->content) == 0) {
         ListElement *tmp = pointer->next;
         pointer->next = pointer->next->next;
         free(tmp);
         success = 1;
      }
      pointer=pointer->next;
   }
   // print out a message whether person was removed or not found.
   if (success == 1) {
      (void)printf("Person removed successfully\n");
   } else {
```

```
        (void)printf("Couldn't find specified Person\n");
    }
}

/*
** function to print out all persons stored in the list.
*/
void show_list(void) {
    ListElement *pointer = &le;
    printf("Erfasste Personen: \n");
            printf("-------------------------------------------------------\n");
    while (pointer->next != &le) {
        pointer = pointer->next;
        print_person(pointer->content);
    }
}

/*
** function to delete all persons stored in the list.
*/
void clear_list(void) {
     ListElement *pointer = &le;
     while (pointer->next != &le) {
        ListElement *tmp = pointer->next;
        pointer->next = pointer->next->next;
        free(tmp);
     }
    (void)printf("List cleared successfully\n");
}
```

Person.h

```
#ifndef PERSON_H
#define PERSON_H

#include <stdio.h>

typedef struct {
    char name[20];
    char firstname[20];
    unsigned age;
} Person;

int compare_person(Person a, Person b);
Person create_person();
void print_person(const Person person);

#endif
```

Person.c

```
#include <string.h>
#include <person.h>

/*
** function to compare if two given persons have the same attribute values.
** @param Person a, Person b
** @return int match
*/
int compare_person(const Person a, const Person b) {
   int match = strcmp(a.name, b.name);
   if (match == 0) {
      match = strcmp(a.firstname, b.firstname);
   }
   if (match == 0) {
      match = a.age - b.age;
   }
   return match;
}

/*
** function to create a new person according to user input.
*/
Person create_person() {
   Person newPerson;
   (void)printf("Name: ");
   (void)scanf("%s", newPerson.name);
   (void)printf("Firstname: ");
   (void)scanf("%s", newPerson.firstname);
   (void)printf("Age: ");
   (void)scanf("%d", &newPerson.age);
   return newPerson;
}

/*
** function to print out a given persons attributes.
** @param Person person
*/
void print_person(const Person person) {
   printf("%s %s, %d\n", person.name, person.firstname, person.age);
}
```