PROGC Praktikum 5 – Verkettete Listen, Dynamische Speicherverwaltung, Modulare Programmierung

Dauer: 4 Lektionen

Aufgabe: Personenverwaltung

Schreiben Sie ein C-Programm, welches eine einfache Personenverwaltung implementiert. Die Daten einer Person sind durch die folgende Struktur definiert:

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

In einer Schleife soll das Programm dem Benutzer jeweils folgende Auswahl bieten, wovon eine Aktion mit Eingabe des entsprechenden Buchstabens ausgelöst wird:

`I(nsert), R(emove), S(how), C(lear), E(nd):`

- Insert: der Benuter wird aufgefordert, eine Person einzugeben
- Remove: der Benutzer wird aufgefordert, die Daten einer zu löschenden Person einzugeben
- Show: eine komplette Liste aller gespeicherten Personen wird in alphabetischer Reihenfolge ausgegeben
- Clear: alle Personen werden gelöscht
- End: das Programm wird beendet

Da wir zur Kompilierzeit nicht wissen, ob 10 oder 10'000 Personen eingegeben werden, wäre es keine gute Idee, im Programm einen statischen Array mit zB 10'000 Personen-Einträgen zu allozieren. Dies wäre ineffizient und bietet zudem grosse Probleme beim sortierten Einfügen von Personen (siehe unten). In solchen Situation arbeitet man deshalb mit dynamischen Datenstrukturen, die zur Laufzeit beliebig (solange Speicher vorhanden ist) wachsen und wieder schrumpfen können. Eine sehr populäre dynamische Datenstruktur ist die verkettete Liste und genau die werden wir in diesem Praktikum verwenden. Dynamische Datenstrukturen werden Sie übrigens im dritten Semester im Modul „Algorithmen & Datenstrukturen“ noch sehr ausführlich behandeln.

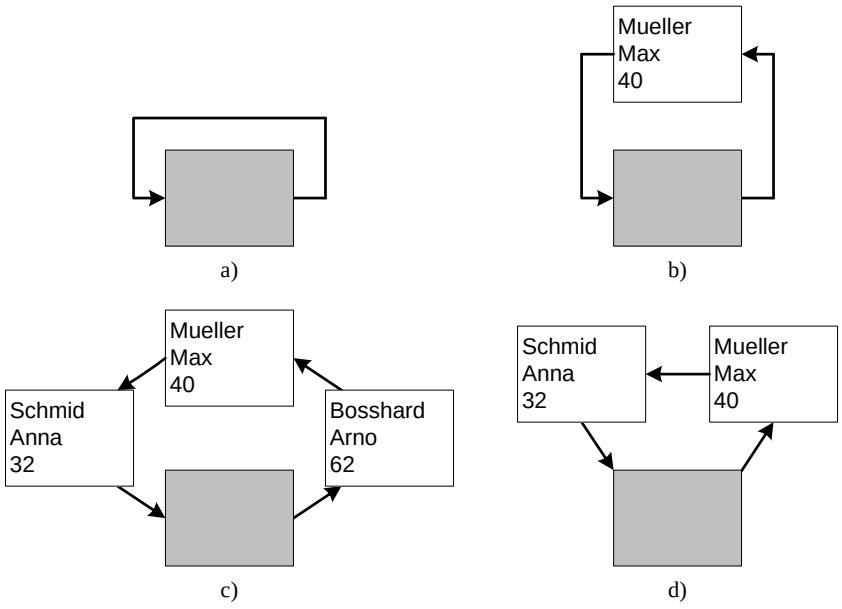
Eine verkettete Liste bedeutet, dass ein Element der Liste einen Datensatz einer Person speichert und zusätzlich einen Pointer auf das nächste Element in der Liste aufweist (siehe Figur 1). In dieser Pointervariablen (`next` in der Struktur unten) steht also einfach die Adresse des nächsten Listenelements. Das letzte Element in der Liste soll wieder auf das erste zeigen, die Liste bildet also einen Kreis. Eine Person kann zugefügt werden, indem dynamisch ein neues Listenelement erzeugt wird und dies in die verkettete Liste eingefügt wird. Beim Einfügen müssen die Adressen der Listenelemente so den Pointern zugewiesen werden, dass nachher wieder ein vollständiger Kreis besteht. Ein Element wird entfernt, indem der Speicher des entsprechenden Listenelements freigegeben und das vorhergehende und nachfolgende Listenelement wieder richtig verknüpft werden, so dass der Kreis wieder geschlossen ist.

Ein einzelnes Listenelement ist durch folgende Struktur definiert:

```
typedef struct LE {
  Person content;        /* In diesem Listenelement gespeicherte Person */
  struct LE *next;       /* Pointer auf das nächstfolgende Element in der Liste */
} ListElement;
```

Diese etwas seltsame Struktur mit einem Namen für die Struktur (`LE`) und nochmals einem Namen mit typedef ist nötig, weil der C-Compiler nur einmal durch den Code und damit auch durch die Struktur geht. Der Name `ListElement` ist erst *nach* dem Durchlaufen der Struktur bekannt und kann deshalb nicht als Typ für das Element `next` in der Struktur verwendet werden, weil er dort noch nicht bekannt ist. Beim Gebrauch können Sie den Namen `LE` ignorieren, arbeiten Sie immer mit `ListElement`.

Die leere Liste besteht aus einem einzelnen Element, welches keine spezifische Person abspeichert und welches auf sich selbst zeigt (Figur 1a). Dieses Element ist der Einstiegspunkt (auch Anker genannt) der Liste und ist das einzige Element, das Sie im Programm direkt kennen und einer Variablen zuweisen. Dieses Element können Sie statisch allozieren (zB `ListElement le;`), denn es existiert während der gesamten Ausführungszeit. Alle anderen Elemente erreichen Sie ausgehend vom Anker, indem Sie einmal, den Pointern folgend, im Kreis herum gehen. Figur 1b zeigt die Liste nach dem Einfügen der Person Max Mueller, 40 Jahre. Nach dem Einfügen von zwei weiteren Personen sieht die Datenstruktur aus wie in Figur 1c. Das Entfernen der Person Arno Bosshard führt zu Figur 1d.



Figur 1: In einer verketteten Liste abgespeicherte Personen

Hinweise:

- Fügen Sie die Elemente alphabetisch aufsteigend in die Liste ein, wobei zuerst der Name, dann der Vorname und dann das Alter verglichen werden. Meier Max kommt also vor Meier Moritz und Mueller Peter, 40 Jahre vor Mueller Peter, 50 Jahre. Dadurch können Sie bei der Ausgabe der Elemente ganz einfach die Personen in der Reihenfolge ausgeben, in welcher sie in der Liste sind.
- Teilen Sie das Programm auf mehrere Dateien auf. Zum Beispiel list.h und list.c um die ListElement-Struktur zu definieren und die entsprechenden Funktionen auf der Liste (insert, remove, show, clear) zu deklarieren und zu definieren; person.h und person.c um dasselbe für die Person-

Struktur und Funktionen auf Personen (zum Beispiel den alphabetischen Vergleich zweier Personen) zu tun; und main.c für das eigentliche Hauptprogramm. Schreiben Sie ebenfalls ein Makefile.

- Achten Sie darauf, allozierten Speicher wieder korrekt freizugeben, wenn Personen aus der Liste entfernt werden.