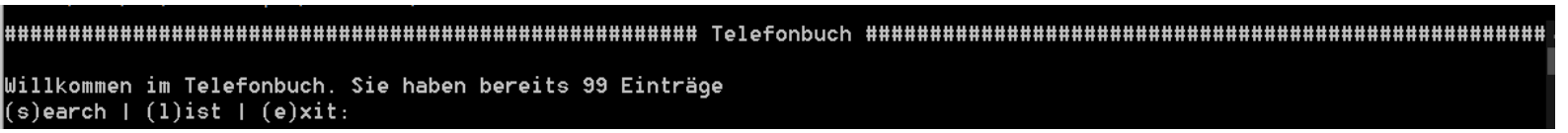
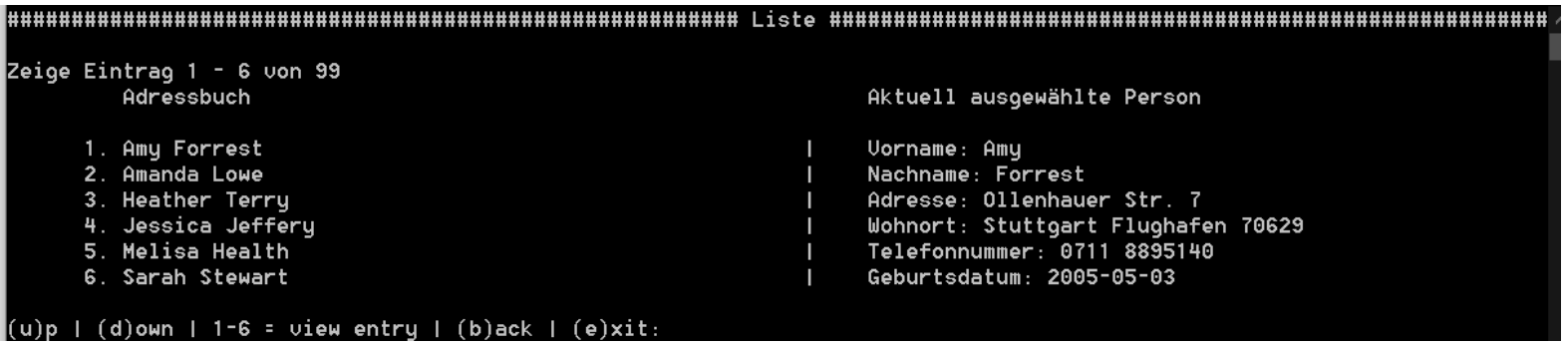
# Anwenderdokumentation

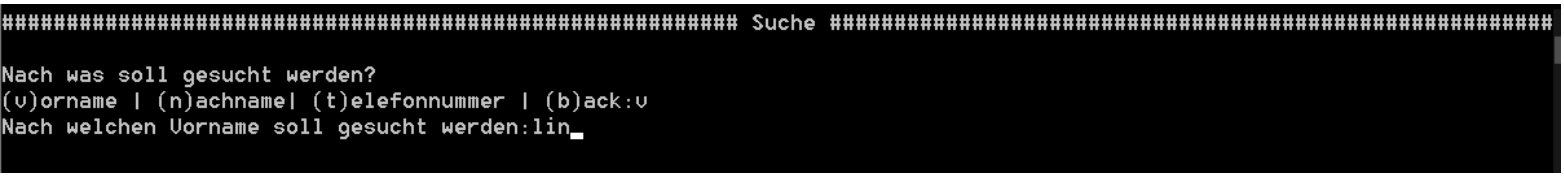
Die Menüs innerhalb des Programmes sind immer gleich aufgebaut.
Der Inhalt befindet sich im oberen Bereich des Bildschirms und die Möglichkeiten zum Navigieren innerhalb des Menüs befindet sich immer als letzte Zeile.




Um den gewünschten Befehl auszuführen gibt man den eingeklammerten Buchstaben ein und bestätigt mit Enter. Beispielsweise, um die gespeicherten Einträge im Telefonbuch anzusehen, müssen wir mit dem Buchstaben „l“ den Menüpunkt „list“ auswählen und bestätigen dies mit Enter.



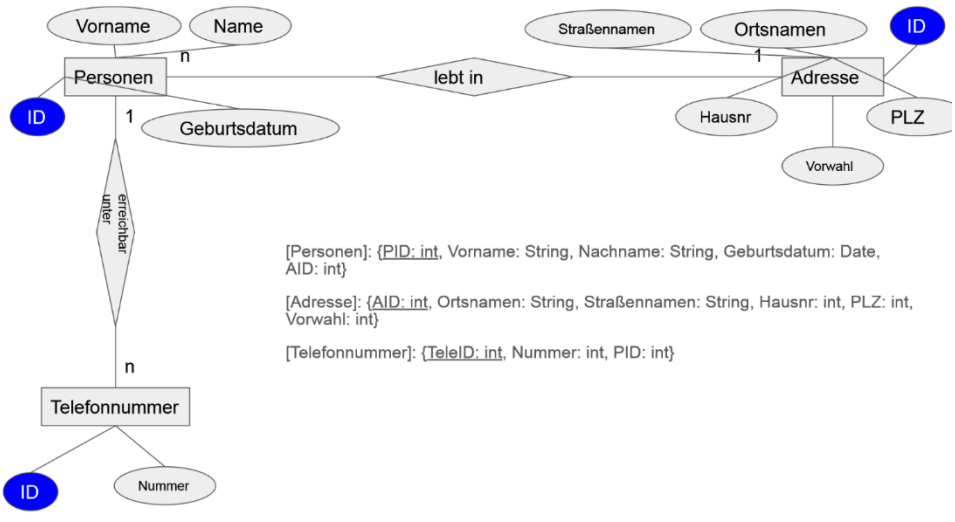

Dies ist die Listenansicht, welche für das Ansehen aller Einträge im Telefonbuch und der Suchergebnisse verwendet wird. Dieses ist in zwei Seiten unterteilt. Auf der linken Seite befinden sich bis zu sechs Einträge welche man mit den Zahlen 1-6 und Enter auswählen kann. Der ausgewählte Eintrag wird dann auf der rechten Seite vollständig angezeigt. Um im Adressbuch die nächsten oder die vorherigen Einträge anzeigen zu lassen verwendet man „d + Enter“ um sich die nächsten Einträge anzuschauen und „u + Enter“ um die vorherigen aufzurufen.
Das Interface bei der Suche sieht folgendermaßen aus:



Bei der Suche kann nach einem Kriterium gesucht werden, bei dem das Suchwort auch nur einen Teil enthalten kann. Das Kriterium ist ebenfalls durch den eingeklammerten Buchstaben + Enter auswählbar. Anschließend wird man nach dem Suchwort gefragt und startet die Suche mit Enter.
Im anschließenden Menü navigiert man wie im Listen Menü durch die Suchergebnisse. Das Programm kann jederzeit mit Schließen des Fensters oder in Menüs mit „(e)xit“ durch „e +Enter“ beendet werden.

# Entwicklerdokumentation

Das Herzstück dieses Programmes ist eine SQLite Datenbank. Diese besteht aus drei Entitäten: Personen, Adressen und Telefonnummern. Alle Entitäten hängen in einer 1-n-Beziehung zusammen. N Personen können in einer Adresse leben und eine Person kann N Telefonnummern haben. Jede Person muss aber in einer Adresse wohnen. In jedem Personeneintrag wird ein Vorname, Nachname, Geburtsdatum und die Adressen-ID gespeichert. Bei Adressen speichern wir Ortsname, Straßenname, Hausnummer, PLZ und Vorwahl. Und in der Telefonnummer-Tabelle werden nur die eigentlichen Nummern und die Personen-ID, zu denen diese gehören, geführt. Für die Beispiel-Daten wurden jeder Person ein zufälliges Geburtsdatum über den Unixepoch zugewiesen. Weiterhin ist die Zuordnung von Person und Adresse auch noch zufällig, genauso wie die Erstellung der Telefonnummer und die Zuordnung zu einer Person. Das läuft aber nur für die Erstellung der Beispiel-Daten.
Für die Integration der SQLite Datenbank nutzen wir eine zusätzliche Bibliothek. Dazu benötigten wir <sqlite3.h>. Diese erlaubte uns aus SQL-Query-Ergebnissen Daten herauszulesen.



Grundsätzlich läuft alles über ein struct Personen, indem die verschiedenen Daten gespeichert werden. Die Telefonnummer und Adresse sind „Unter-Structs“ in dem Personen-Struct. Abhängig vom gewünschten Index aus der Datenbank wird dieses Struct nun in personausSQL mit Hilfe einer SQL-Query gefüllt. Zur Darstellung im Fenster wird ein Array von sechs Personen-Structs nun genutzt. Grundsätzlich wird dieses erstmal mit den ersten sechs Indizes gefüllt. Zum Scrolling verschieben wir jedes Arrayelement um sechs Indizes nach oben oder unten. Bei den beiden Randfällen werden, je nach Fall entweder das erste Element auf den minimalen Index oder das sechste Element auf den maximalen Index gesetzt. Das ist zwar nicht die eleganteste Lösung, aber diese ist dafür sehr stabil. Bei einer Suche wird dasselbe Prinzip angewendet, nur dass jetzt nicht durch die Indizes der Datenbank gescrollt wird, sondern durch ein Array der Ergebnis-Indizes. Sollten jedoch weniger als sechs Ergebnisse kommen, werden die überflüssigen Structs mit ID = -1 gesetzt und mit dieser Flag bei der Ausgabe ignoriert.

Bei der UI wird das Interface über eine Breite von 120 Zeichen ausgelegt, an der sich jedes UI Element orientiert. Einen Großteil der UI wird dynamisch auf Basis von den Datenbankergebnissen generiert.
Die Titel werden zentral gesetzt, indem die Anzahl an Rauten in zwei Grundschritten berechnet wird.
Dabei wird zuerst der linke Teil berechnet, bei dem die Hälfte der Titellänge plus einem Leerzeichen von 60 subtrahiert wird. Diese werden dann in die Zeile geschrieben und der Titel wird, umschlossen mit den Leerzeichen eingefügt. Das Gleiche wird auch auf der rechten Seite gemacht, allerdings wird zusätzlich noch das Modulo von 2 des Titels subtrahiert, um Titel zu berücksichtigen, die nicht restlos durch zwei teilbar sind.
bei dem Listenbildschirm werden beide Seiten der UI, aufgeteilt durch das zentral gesetzte „|“ in einer Funktion generiert.
Die Parameter geben dabei an, welcher der sechs Einträge gerade generiert wird und was auf der rechten Seite angezeigt werden soll.
Den linken Teil, also das Adressbuch wird durch das „Structarray“ befüllt.
Die Abstände durch Leerzeichen zu den andern UI Elementen werden auch nahezu vollständig durch Berechnungen anhand der Länge der Angaben kalkuliert und schrittweise hinzugefügt, bis eine vollständige Zeile erstellt wurde. Dieser Ablauf wird sechs Mal wiederholt, um die vollständige Liste anzuzeigen.

## Fehlende Features

Leider fehlt vollständig die Bearbeitung der Datenbank aus C. Es ist somit leider nicht möglich Daten hinzuzufügen, zu editieren oder zu löschen. Angedacht waren diese auch auf Basis der Personen-Structs. Das Hinzufügen wäre über ein SQL-Query gelaufen, indem man im Hauptmenü einen neuen Eintrag anstoßen kann und das Programm Schrittweise durch die Erstellung leitet und währendessen ein neues Personen Struct erstellt und dann durch eine Import Funktion in die Datenbank eingepflegt wird. Durch die Struktur der Datenbank wären selbst unvollständige Angaben kein Problem gewesen, wenn zumindest eine Adresse angegeben gewesen wäre. Auch das Löschen und das Editieren wäre über parametrisierte SQL-Queries möglich gewesen, welche über die Listenmenüs in der Ansicht des gesamten Telefonbuchs oder auch bei den Suchergebnis auf den angewählten Eintrag anwendbar gemacht werden soll. Bei dem Editieren wäre als Basis einen Teil der Logik zum Erstellen eines Eintrages auf UI Ebene wiederverwendet, bei der nur die vorherigen Werte mit aufgelistet werden und auf Datenbankebene ein Update auf die Personen-ID erfolgen würde. Ein Entfernen des Eintrages wäre durch einen „delete“ Befehl auf die Personen-ID umgesetzt worden.

## Lessons learned

Theo:

Ich für meinen Teil habe die Komplexität der Anbindung an die Datenbank aus C massiv unterschätzt. Leider waren viele Hilfen im Internet mit C# oder C++ geschrieben. Durch das fehlende Verständnis meinerseits hat die Fehlersuche zu Beginn sehr lange gedauert, wodurch ich nur sehr langsam voran kam ich nur sehr langsam voran.

Durch das Programmieren in einer nicht objektorientierten Sprache sind leider auch einige Möglichkeiten weggefallen, die man in Java beispielsweise hatte. Viele Dinge konnte man aber auch ohne Probleme mit Structs lösen.

Die eigentliche Arbeit an der Datenbank hat jedoch relativ gut funktioniert. Zwar war es ein neues SQL-System, aber SQL bleibt am Ende SQL.

Für die nächsten Projekte muss sich mein Zeitmanagement deutlich verbessern. Die Arbeitsaufteilung sowie die Kommunikation liefen hingegen sehr gut.

Fabian:

Bei mir habe ich ebenfalls den Aufwand der Erstellung der UI unterschätzt, dadurch, dass ich zuerst geplant hatte eine Bibliothek für die UI zu verwenden (ncurses), welche sich nach weiteren forschen als unpraktikabel unter Windows herausstellte und somit die Platzierung der UI Elemente durch Koordinaten nicht mehr möglich machte. Ebenfalls bin ich davon ausgegangen, dass viele der Komfortfunktionen, welche ich unter anderen in C#, Kotlin und Python zur Verfügung hatte, ebenfalls nutzen konnte, was jedoch nicht der Fall war. Als Beispiel hierbei nenne ich das einfache Ändern des Datentyps einer Variablen, sicheres Typecasting hauptsächlich von int zu char * oder auch die einfache Bearbeitung eines Strings mit Funktionen wie „drop last/first n char“ oder auch das Ersetzen von char-Abfolgen durch andere char-Abfolgen. Auch hat die Wahl von Visual Studio Code, als Ausweichmöglichkeit von Visual Studio durch mangelnde Unterstützung von reinem C, auf meiner Seite, zu anfänglichen Problemen bei der Kompilierung geführt. Dabei war das Problem, dass die von Microsoft zur Verfügung gestellten Konfigurationsdateien erst von Hand bearbeitet werden müssen und diese keinen Compiler oder Debugger durch die Erweiterungen mitliefert. Dadurch mussten diese nachinstalliert werden. Sobald dies aber getan wurde hat es den Workflow von mir sehr verbessert, da ich mit VS Code bereits im Grunde vertraut war.

Als Fazit ziehe ich daraus, dass das Wegfallen einer UI Bibliothek mir deutlich gezeigt hat, wie ich mit Strings oder in dem Fall char * umgehen muss und wie ich alternative Möglichkeiten für Komfortfeatures aus anderen Sprachen erstelle. Dadurch habe ich ein besseres Verständnis der Sprache C bekommen und schätze damit zusätzlich die Arbeit, die in andere Sprachen bereits von mir genommen werden. Die Kommunikation und die Arbeitsaufteilung war sehr gut, bei der auch jeder von uns seine Stärken gut in das Gesamtprojekt einbringen konnte.