Belegarbeit Labor 5 - Grundlagen der Programmierung HTW

Lennard Wittenberg

4. März 2024

Zusammenfassung

Belegarbeit zum Laborblatt 5. Es enthält eine Beschreibung der Aufagbenstellung, Grundenlagen, Vorgehen und Lösungen zu den Laboraufgaben und ein Fazit.

Inhaltsverzeichnis

1 Einführung

Im Rahmen des HTW Moduls CE22 ”Grundlagen der Programmierung Labor-5”wird in dieser Belegarbeit die Verwendung von Verkettete Listen und der Multimedia-Bibliothek SDL2 vertieft. Die konkreten Aufgabenstellungen, erforderlichen Grundlagen, mein Vorgehen bei den Problemstellungen sowie die letztendlichen Lösungen werden in den folgenden Abschnitten detailliert beschrieben. Laborblatt 5

1.1 Aufgabenstellung

Das 5.Labor beinhaltet 3 Aufagben.
1.) In der ersten Aufgabe sollen grundlegende Fragen zu den Themen des Labors beantwortet werden. Dies dient zum einen zur Wiederholung einiger vorgegangen Grundlagen, als auch zur Aneignung der neuen Grundlagen.
2.) In der zweiten Aufgabe wird der Umgang mit Verketteten Listen vertieft. Es soll ein Projekt erstellt werden, mit dem Namen nach bestimmten Kritierien sortiert werden können. Das Projekt soll universell sein. Daher werden die zu sortierenden Namen aus einer CSV-Datei ausgelesen.
3.) In der dritten Aufgabe wird soll die Verwendung von SDL2 eingeführt werden. In dem zu entwickelndem Projekt werden Grundlagen der Objekt-orientierten Programmierung behandelt.

1.2 Aufbau der Arbeit

In diesem Abschnitt soll der Fokus auf den Aufbau der Belegarbeit gelegt werden und die einzelnen Abschnitte sollen erläutert werden. Der verbleibende Teil der Belegarbeit ist in vier Abschnitte unterteilt. Im ersten Abschnitt finden Sie alle erforderlichen und hilfreichen Grundlagen und Informationen, um die Laboraufgaben zu lösen und ein allgemeines Verständnis für die in dieser Belegarbeit behandelten Themen zu entwickeln.
Im zweiten Abschnitt werden in jeweils einem Unterabschnitt meine Vorgehensweise zur Bearbeitung der Laboraufgaben erläutert. Dabei werden anhand von Beispielen und Codeauszügen die Lösungen präsentiert. Zum Abschluss erfolgt im dritten Abschnitt eine zusammenfassende Schlussbetrachtung im Kontext der behandelten Thematiken dieser Belegarbeit. Im vierten Abschnitt finden Sie anschlieSSend ein Literaturverzeichnis.

2 Grundlagen

Eine vollständige Auflistung aller benötigten Grundlagen ist in diesem Verzeichnis zu finden.
Die wichtigsten Themen sind CSV-Dateien, verkettete Listen und die Multimedia-Bibliothek SDL2.

3 Vorgehensweise

In den folgenden Textabschnitten werden wie in **??** beschrieben meine Vorgehensweise bei der Bearbeitung der Laboraufgaben dargestellt.

3.1 Vornamenrangliste

Ziel: Erstellung eines Projektes mit dem Namen nach bestimmten Kritierien sortiert werden können. Das Projekt soll fünf Krieterien erfüllen.
1. Das Programm nimmt als Argument den Namen einer beliebigen CSV-Datei, welche dem vorgebenen Format der Namensliste folgt, und die darin gespeicherten Namen aus.
2. Standardmäßig soll die Top 10 der Namen sortiert nach der Anzahl der Namensträgern ausgegeben werden.
3. Dem Nutzer soll ermöglicht werden selber auszuwählen, wie großdie Top X sein soll. Also wie viele Namen ausgegeben werden sollen.
4. Das Programm soll über eine Option verfügen, die alle Einträge nach Namen sortiert ausgibt.
5. Das Programm soll über eine Option verfügen, die die Ausgabe nach Geschlecht filtert.
Wie dieses Projekt verwendet werden kann, geht aus der Commandarguments-Datei und dem Laborblatt 5 hervor.

3.1.1 Auslesen der CSV-Datei und Verbinden des Codes (Schnittstelle)

Diesen Punkt hatte ich als letztes bearbeitet, daher musste ich bei der Implementation darauf achten, dass alles mit meiner bisherigen Arbeit kompatibel ist. Zunächst mussten einige grundlegende Funktionen eingefügt werden. Zum einem die Struktur, welche die Elemente der Liste darstellen soll. Darüber hinaus die drei Funktionen, die die Liste nutzbare machen. Eine Funktion zum hinzufügen von neuen Elementen, eine zum ausgeben der Elemente und eine Funktion die die Größe der Liste zurückgibt. Die Länge von Listen wird häufig bei anderen Funktionen verwendent. In meinem Projekt heißen diese addHead(), size und printLinkedList. Bei letzter handelt es sich um das Debugging tool des Authors. Die genauen Definitionen und Deklarationen der Funktionen sind in meinem Repository zu finden.(Verlinkung im späteren Verlauf) Ungefähr 75 Prozent des verwendeten Code stammt aus Refactoring von meinen älteren Projekten und Java Projekten. siehe Dafür das functions Verzeichnis und das Unterverzeichnis List Um die Daten aus den CSV-Datein verwenden zu können, sind drei Schritte nötig. 1. die CSV-Datei muss importiert werden, 2. Die in der Datei gespeicherten Daten müssen extrahiert werden und 3. werden die Daten über Variablen verwaltet, um neue Listen Elemente zu erschaffen. Dafür wird die Funktion addHead() verwendet und printLinkedList() als Debugging tool. Der erste und zweite Schritt werden in der main Funktion ausgeführt.

```
int main(int argc, char **argv){
        const char *filename = argv[1]; // vollständiger Name der CSV Datei
        FILE *csv; // Speichert den gesamten Inhalt der CSV_Datei temporär.
```

```
        char buffer[1024]; // Arrays welches die Zeile der Datei speichert,
                           // die gerade verarbeitet wird.
        int rank = 1;          //@param rank stellt einen Mitglied der Liste dar.
                        //Durch dieses Mitglied kann nachvollzogen werden,
                        //in welcher Reihenfolge die Elemente hinzu gefügt wurden.
        struct cmdargs options; // Command handeling
        struct Node *head = malloc(sizeof(struct Node));
        cmdargs_parse(&options, argc, argv);
        head->rank = 0; // head ist der erste Element jeder Liste.
                        // In diesem Fall ist head leer und dient als Leerzeile
        head->frist_name = "";
        head->gender = ' ';
        head->anzahl = 0;
        head->next = NULL;
        // die folgenden f-Funktionen oder file Funktionen wurden in Vorlesung acht
        // behandelt. mit dem man (kurz für manuell) Command kann nachgeschlagen werden,
        //  wie die jeweilige Funktion funktioniert.
        csv = fopen(filename, "r");
        // error handeling für den Fall, dass keine valide Datei übergeben wurde.
        if(csv == NULL){
                fprintf(stderr, "Couldn't open '%s'; %s\n", filename, strerror(errno));
                return 1;
        }
        else{
                fgets(buffer,1024,csv);
                printf("%s", buffer); // diese Zeile druckt die Kopfzeile der CSV-Datei.
                while(fgets(buffer,1024,csv)){
                        parse_name(head,buffer,rank);// solange es noch nicht verarbeite
                        // Zeilen in der Datei gibt, sollen diese im dritten Schritt
                        // mit einer Hilfs Funktion verarbeitet werden.
                        rank++;
                }
                fclose(csv);
        }
        check_options(options,head); Verwaltung der Programm Optionen
return 0;
}
```

Die Verwaltung der Programm Commandos läuft über das Projekt cmdargs, welches bereits in einem früheren Labor etabliert wurde. Der dritte Schritt und das übertragen in die Liste findet in dieser Funktion statt:

```
void parse_name(struct Node* head, char* line, int rank){
        char* name;
        char* gender;
```

```
        char* count;
        // @params name, gender, count sind Puffer Variablen welche temporär
        // die Daten der CSV-Datei speichern,
        int anzahl = 0; // da die Daten nur als Strings empfangen werden können
                        // muss eine zweite Variable für das Mitglied "anzahl"
                        // erzeugt werden
        int rc = 0;
        // sscanf wurde in Vorlesung acht behandelt und wird benutzt, um die
        // Daten der Zeilen in den Variablen zu speichern.
        rc = sscanf(line, "%m[^,],%m[^,],%m[^,]", &name, &count, &gender);
        anzahl = argparse_int(count);
        // Überprüfung, ob die Umwandlung erfolgreich war.
        if(rc != 3){
                return;
        }
        // Diese Funktion erstellt aus den Variablen, die die Daten der einzelnen
        // Zeilen enthalten, die neuen Listen Elemente.
        addName(head,name,gender[0],rank,anzahl);
}
```

3.1.2 Sortieren nach Namen-alphabetisch

Die Option besteht aus drei Teilen. Einer Hauptfunktion, welche die Liste sortiert und eine Hilfsfunktion, die die neue Liste auf die Konsole druckt und die benötigten Parameter an die Hauptfunktion weiterleitet. Die dritte Hilfsfunktion ist dafür zuständig zwei Elemente in einer Liste miteinander zu vertauschen. Da die Nebenfunktion keine Signifikanz für den eigentlichen Erfolg der Option hat, gehe ich hier nicht weiter auf den Code dieser Funktion ein. Um die Element alphabetisch zu sortieren verwende ich eine abgewandelte Form des selection-sorts. Bei diesem wird ein Speichermedium mit Hilfe von zwei Schleifen immer wieder durchlaufen und nach dem gewählten Kriterium sortiert. Dabei wird das erste Element mit dem Element in der Liste vertauscht, welches in der alphabetischen Rangfolge am höchsten steht. Dafür wird die zweite Hilfsfunktion verwendet. Die Überprüfung des Kriterums wird mit der Standard-Funktion strcmp() aus string.h durchgeführt, wobei die zu überprüfenden Strings die Mitglieder ”frist_name”der Listen Elemente sind. Hier der wichtigste Teil des Sortieralgorithmuses:

```
tmp = head_ref;
                int swapped = 0;
                for(int j = 0; j< count -1 - i; j++){
                        struct Node* n1 = *tmp;
                        struct Node* n2 = n1->next;
                        if(strcmp(n1->frist_name,n2->frist_name)>0){
                                *tmp = swap(n1,n2);
                                swapped = 1;
```

```
}
tmp = &(*tmp)->next;
```

Den vollständigen Code der Funktionen sind in der Datei List_func.c zu finden. Die Hauptfunktion von Zeile 85 bis 105, Die Tauschfunktion swap() von Zeile 78 bis 83 und die Schnittstelle alpha() von 142 bis 150.

3.1.3 Ausgabe der Top 10 und Top X

Diese Option ist prinzip vollkommen identisch zur Option Sortieren nach Namen. Die einzigen Unterschieden sind, dass ein anderes Mitglied änzahl"überprüft wird. anzahl steht hier die Anzahl aller Namensträger mit dem selben Vornamen, dass Top X die Standard Ausgabe des Programms ist und dass für die Überprüfung des Kriteriums keine Funktion wie strcmp() benötigt wird. Es handelt sich also lediglich um kleine Codeänderungen. Der Code ist ebenfalls in der Datei List_func.c findbar.

3.1.4 Sortieren nach Geschlecht

Die Elemente der Liste sollen nach ihrem Struktur Mitglied gender(Geschlecht) sortiert werden. Das Mitglied hat den Datypen char. In diesem Projekt wird sich nur auf das biologische Geschlecht bezogen. Dadurch ist das sortieren wesentlich leichter. Ich habe das Problem so gelöst, dass die Liste in zwei Durchgängen von Anfang bis Ende durchlaufen wird. Bei diesen Durchläufen wird überprüft ob dem Listen Element das weibliche oder männliche Geschlecht zugewiesen ist. Entspricht der Wert des jeweiligen Elements dem gesuchten Geschlechts werden alle Informationen des Elements auf der Konsole ausgegeben. Zuerst wird nach den weiblichen Namensträgern gesucht und dann den männlichen. Den vollständigen Code der Funktionen ist in dieser Datei zufinden von Zeile 48 bis 63. Diese Option kann mittels –gender_sort ausgeführt werden. main-Datei

3.2 HelloSDL, Version 2

Ziel: Fortführen eines Projektes, welches mittels der SDL2 Umgebung ein Object in einem Fenster kontrollierbar macht.
1. Sobald das Objekt eine Grenze des Fensters berührt, soll dieses mittles eines elastischen Stoßes umgekehrt werden.
2. Klickt der Nutzer auf eine leere Postion im Fenster, wird ein neues Objekt erzeugt.
3. Objekt im Fenster sollen in vertiakle Richtung beschleunigt werden, wenn es vom Nutzer angeklickt wird.
4. Objekte die nicht mehr hüpfer sollen zum Stillstand begracht werden.
5. Objekte die sich im Stillstand befinden, werden nach einer Sekunde entfernt.

3.2.1 elastischer Stroß

In der Main-Datei sind die Grenzen und die Maße der Objekte definiert. Sobald eine Seite eines der Objekte (Quadrat) eine Grenze des Fensters berührt, muss die Geschwindigkeit des Objektes umgekehrt werden. Wird eine Grenze auf der x-Achse berührt, also x-maximal oder x-minimal, wird die Geschwindigkeit in x-Richtung invertiert. Gilt auch in y-Richtung. Es ist jedoch zu berücksichtigen, dass das Objekt nicht erst abgestoßen wird, wenn seine Koordinaten die Grenze erreicht, sondern die Kante des Quadrats. Deshalb müssen bei der Überprüfung, ob die Grenze erreicht wurde, die Maße des Objekts von den Grenzpunkten subtrahiert werden. hier ein Beispiel:

```
// Beispiel für den Fall, dass das Objekt die rechte Grenze des Fensters
// erreicht
if(obj_sq.x > 590){
        obj_sq.speedx*=-1;
        obj_sq.dead = 0;
```

3.2.2 Erzeugung neuer Objekte

Dieses Unterprojekt habe ich nicht geschafft. Da es für diesen Unterpunkt eine Anleitung gab, kann ich selber keine konkreten Probleme nennen. Ich bin bei der sechsten Teilaufgabe dieses Unterpunktes stecken geblieben und konnte es nicht lösen.

3.2.3 Beschleunigung per Klick

Auch dieses Unterprojekt habe ich nicht beenden können. Allerdings kann ich zu diesem Punkt mehr sagen.
Ziel: Das Objekt soll, wenn es angeklickt wurde konstant in y-Richtung einmalig beschleunigt werden.
Zur Beschleunigung kann eine sehr simple Funktion geschreiben werden.

```
void boost(struct game_objects* obj){
        obj->speedy -= 25;
}
```

Um zu ermittlen, ob ein Objekt angeklickt wurde, werden fünf Dinge gebraucht. erstens eine Überprüfung, ob der Nutzer während das Program läuft Befehle eingegeben hat. In SDL heißt das SDL_PollEvent(). Diese Funktionialität ist bereits vorhanden bei dem zur Verfügumg gestelltem Projekt. zweitens eine Überprüfung ob eine Maustaste gedrückt wurde. SDL stellt zu diesem Zweck SDL_MOUSEBUTTONDOWN zur Verfügung. Hier kurz die Anmerkung, dass diese und alle anderen Bedigungen Teil einer switch Struktur sind. Zusammen mit dem dritten und vierten Teil SDL_GetMouseState und SDL_BUTTON() kann überprüft werden, wo und welche Maustaste gedrückt wurde. Mir ist dieser Teil leider nicht gelungen. Daher kann ich keinen Lösungsweg bereit stellen. Fünftes ist eine Überprüfung nötig, um zu prüfen, ob der Mausklick innerhalb

des Quadrates gemacht wurde. Meine Idee war es zu testen, ob der Klick 25 Längeneinheiten mehr oder weniger als die x- und y-Koordinaten des Quadrats erfolgt ist. Sollte das der Fall sein, sollte das Objekt beschleunigt werden. Der Fehlerhafte Code hierzu ist in dieser Datei zu finden.

3.2.4 Stillstand

Das Objekt hüpft dann nicht mehr, wenn die Geschwindigkeit in y-Richtung endweder konstant Null oder konstant fallend ist. Da allerdings y Grenze für den Stoßnicht die Grenze des Fensters ist. (wurde so vorgegeben) Habe ich die Bedingung so gewählt, dass das Objekt zum Stillstand gebracht werden sollen, solbald es diese Grenze passiert.

```
// @param dead ist eine eingens dafür geschaffenes Mitglied der
// Objekt Struktur
// Sobald die Grenze überschritten wird bewegt sich das Objekt nicht mehr
// und die Lösung wird eingeleitet
        if(obj_sq.dead == 0){
                obj_sq.x += obj_sq.speedx;
                obj_sq.y += obj_sq.speedy;
        }
        // Code dazwischen ....
        if(obj_sq.y >= 400){
                obj_sq.speedy *= -1;
                obj_sq.dead = 0;
        // Code dazwischen ....
        }
        if(obj_sq.y >= 402){
                obj_sq.speedy = 0;
                obj_sq.speedx = 0;
                obj_sq.dead = 1;
        }
```

3.2.5 Entfernung von Objekten

Um das Objekt zu entfernen benutzte ich die free() Funktion aus dem Pointer Kapitel der Vorlesungen. Für den Timer von einer Sekunde habe ich diesen Code benutzt:

```
        if(state->obj.dead == 1){
                int check = 0;
                struct timespec start, finish;
                while(check < 1){
                        clock_gettime(CLOCK_MONOTONIC,&start);
                        sleep(1);
                        clock_gettime(CLOCK_MONOTONIC,&finish);
```

```
                        check = abs(finish.tv_sec - start.tv_sec);
                }
                if(check >= 1){
                        free(state);
                }
        }
}
```

Gesamtes Projekt

4 Fazit

Das fünfte Labor verfehlt meiner Ansicht nach seinen Zweck. Die ursprüngliche Abgabe dieses Labors war knapp vor der Klausur. Da mit diesem Labor Listen in die praktische Arbeit aufgenommen wurden, war es die einzige und zu kurzfristige praktische Übung für Listen vor der Klausur. Zudem wurden Listen in diesem Labor zu spezifisch genutzt. Im Prinzip wurden nur die bereits bekannten Sortieralgorithmen mit einer neuen Datenspeicherstruktur geübt. Zudem war das Inkludieren von Datensatzverarbeitung in Verbindung mit CSV-Dateien unnötig. Entweder hätten diese separiert oder zu verschiedenen Zeitpunkten eingeführt werden sollen. Was mich zurück zum Labor vier bringt. Meiner Meinung nach wurde zu lange mit dem gcd Projekt gearbeitet. Diese fehlende Zeit ist nun in späteren Laboren spürbar. Wäre dieses Labor meine erste Erfahrung mit Listen gewesen, hätte ich erheblich länger für die Bearbeitung gebraucht.
Im vierten Labor wurde das Projekt hellosdl2 vorgestellt und es wurde sehr knapp damit gearbeitet. Retrospektiv getrachtet empfinde ich den Einstieg im vierten Labor als unnötig, gerade weil die Änderungen im fünften Labor sofort wieder verworfen werden. Daher sehe ich die Aufgabe HelloSDL, Version 2 als eigentlichen Einstieg in das Thema und als erstes Projekt scheitert es auf ganzer Strecke. Wenn man etwas Neues wie zum Beispiel eine neue Entwicklungsumgebung lernt, ist es meiner Meinung nach niemals ein guter Einstieg, wenn man an einem Projekt weiterarbeitet, das von jemand anderem geschrieben wurde. Allein den Code ohne Wissen darüber zu verstehen ist schon eine Aufgabe an sich. Kommen dann noch komplexe Strukturen, Aufgabe, Error und Event-Handling dazu, dann ist es ein Rezept für ein Desaster. Nachdem ich mich an diesem Projekt abgemüht habe, weißich genau so viel wie vor dem Projekt. Alles in allem hatte ich mir mehr von der letzten Laborübung erhofft.

5 Literaturverzeichnis

Geeks for Geeks
Stack Overflow
Vergangene Projekte
Leifi-Physik