Belegarbeit Labor 3 - Grundlagen der Programmierung HTW

Lennard Wittenberg

3. Dezember 2023

Zusammenfassung

Belegarbeit zum Laborblatt 3. 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-3”beschäftige ich mich in dieser Belegarbeit mit dem Testen und Dokumentieren von Programmen anhand einiger kleiner Beispiel-Funktionen. Zusätzlich werde ich weitere Nutzungsmöglichkeiten des Build-Systems ”make”thematisieren. Die konkreten Aufgabenstellungen, erforderlichen Grundlagen, mein Vorgehen bei den Problemstellungen sowie die letztendlichen Lösungen werden in den folgenden Abschnitten detailliert beschrieben. Laborblatt 3

1.1 Aufgabenstellung

Das 3.Labor beinhaltet 6 Aufagben.
1.) In der ersten Aufgaben 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.) ReadMe aktualisieren. In dieser Aufgabe wird die ReadMe Datei des Git-Lab Projekts erweitert. Die Datei soll dem allgemeinen Standard von ReadMe Dateien angenährt bezeihungsweise angepasst werden.
3.) Makefile mit Unit-Tests erstellen. Ziel dieser Aufgabe ist es den Umgang mit ”makeßu verbesseren. Dazu sollen drei Quelltexte und eine Begleit-Datei (Header) ertsellt beziehungsweise verbessert werden. Zusätzlich wird eine makefile für einen einfacheren Umgang gefordert. Mit diesen Dateien soll eine Funktion äddërstellt, getestet und dokumentiert werden.
4.) Subtraktion. Analog zu Aufgabe 3 wird in dieser Aufgabe eine Funktion ßub”, die die Differenz zweier Zahlen bestimmen soll, erstellt, getestet und dokumentiert.
5.) GröSSter gemeinsamer Teiler. Auch in dieser Aufgabe wird eine Funktion ”gcd”, die den GröSSten gemeinsamen Teiler bestimmen kann, analog zu 3.) und 4.) bearbeitet. Zusätzlich werden in dieser Aufgabe flags eingeführt. Diese ermöglichen es dem Nutzer zusätzliche Answeisungen zu übermittlen. Im diesem Fall handelt es um andere Funktionen, die die Ausgabe des main Programmes verändern.
6.) Mysort. Analog zu Aufgabe 5.) soll ein Projekt bestehend aus Quelltexten erstellt werden, das eine Funktion zum sortieren eines Arrays erstellt, testet und dokumentiert. Genauere Details zu den einzelnen Aufgaben, Grundlagen sowie den verwendeten Programmen zum Beispiel doxygen finden Sie in den Abschnitten Grundlagen und Vorgehensweise.

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 all-

gemeines 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

Die erforderlichen Grundlagen zur erfolgreichen Bearbeitung des Laborblattes 3 waren wie in 1.1 erwähnt in Aufgabe eins gestellte Fragen. Diese werden in diesem Textabschnitt beantwortet.
In Labor 3 wird der Umgang mit verschiedenen Systemen und Begriffen gefordet, um diese zu verstehen und die Aufgaben bearbeiten zu können, müssen diese ersten mal geklärt werden. Zuerst ist wichtig zu verstehen, was ein README und eine Pipeline-Badge sind, da diese in Git-Lab eine wichtige Rolle spielen. Ein README ist eine Textdatei, die zur Einführung in ein Projekt verwendet wird. Das naheliegendste Beispiel wäre hier das GitLab-Repository, in dem sich diese Datei befindet. In dieser Einführung werden standardmäSSig alle Inhalte eines beliebigen Projekts kurz zusammengefasst und vorgestellt. Diese werden auch oft mit einer Pipeline-Badge versehen. Eine Pipeline-Badge bietet eine Möglichkeit, Teilinformationen über ein Projekt darzustellen. Eine Badge besteht aus einem kleinen Bild (meistens ein Symbol für einen bestimmten Status) und einer URL. Die URL verweist dabei auf das Bild. Badges werden in Git-Lab unter den Projektbeschreibungen angezeigt.
Ein großer Bestandteil der Belegarbeit sowie des Labors sind "Companion-Include-Dateien". Daher folgt auch hier ein kurzer Auszug zum Thema Companion-Dateien:
Was ist eine Companion-Include-Datei und warum sollte Sie inkludiert werden? Eine Companion-Datei ist eine zugehörige Header-Datei zu einer C-Datei. Das selbe gilt auch anders herum. Allerdings gibt es nicht zu jeder C-Datei auch eine Companion-Datei. Zum Beispiel core.c gehört zu core.h und core.h gehört zu core.c
Außerdem wenn eine C-Datei 1 (Header-Datei) Funktionen initialisiert, die in der C-Datei 2 verwendet werden sollen, Muss C-Datei 2 die C-Datei 1 inkludieren, um die Funktionen zu deklarieren und nutzen zu können. Beispiel: printsincos.c und sincos.c benötigen printsincos.h.
Der Hauptbestantteil des Labors ist der Umgang mit dem Build-System Make, dem Testen von Funktionen und Programmen sowie der Dokumentation von Programmen. Daher müssen die verwendeten und Programme und deren Verwendung kurz erläuetert werden. Make ist ein Build-System in Form einer Textdatei. in der Regel wird Make angewendet, um das Erzeugen, Löschen oder Kompilieren von einer oder häufiger mehrerer Quelltext- bezeihungsweise Objekt-Dateien in einem Projekt zu vereinfachen. Man erzeugt eine solche Datei im dem Ordner des Projekts mit dem bash-Kommando:

```
nano makefile.
```

Der Vorteil einer Make-Datei ist, dass die Answeisungen zum kompilieren oder erzeugen nur einnmal in der make-datei definiert werden müssen. AnschlieSSend kann man diese Befehle einfacher und schneller ausführen. Diese Befehle nennt man Regeln.
Eine Regel beinhaltet den Code um beim Aufruf die geforderten Dateien zu erzeugen, löschen oder kompilieren. Ein Beispiel wäre eine Regel zum kompilieren beliebiger Quelltexte in eine Objektdatei. Eine Regel besteht aus einem Targetnamen, einer Auflistung der benötigten Dateien und den Anweisungen, die durchgeführt werden sollen.
Beispiele für eine vollständige Regel:

```
Targetname: Auflistung der benötigten Dateien
        Anweisungen
```

```
Name: dateiname1.c dateiname1.h
        gcc -Wall -Werror dateiname1.c -o dateiname1
```

Man kann in einer Make-Datei beliebig viele Regeln aufstellen. Beispiel eines Aufrufs:

```
make Name
```

Wichtig ist Außerdem, Wie man aus Objektdateien eine ausführbare Dateien erzeugt. Der Linker von C bildet mittels der Objektdateien und Bibliotheken ein ausführbares Programm. Beispiel für gcd:

```
gcc gcd.o gcd_functions.o -ogcd
```

Zum Testen der Funktionen sollen in diesem Fall Unit-Tests und Test-Coverage verwendet werden. Unit-Tests sind ein wichtiger Bestandteil der Softwareentwicklung, insbesondere in der objektorientierten Programmierung. Sie ermöglichen es, kleine Teile des Codes (die sogenannten Units) isoliert zu testen und somit Fehler schneller und einfacher zu finden. Die Test-Funktionen können über die Header-Datei ässert.hïn eine Programm eingebunden werden. Assert Beispiele:

```
Assert.AreEqual(), AssertIsTrue(), Assert.IsNotNull()
```

Beim Testen von Funktionen sollte man darüber hinaus die Funktion exit() nicht verwenden. Diese funktioniert folgendermassen:
1. Löscht nicht geschriebene gepufferte Daten.
2. Schließt alle geöffneten Dateien.
3. Entfernt temporäre Dateien.
4. Gibt einen ganzzahligen Exit-Status an das Betriebssystem zurück.
Diesen Exit-Status zu überprüfen ist kompliziert und müsste für jeden einzelnen Test einzeln vorgenommen werden. Daher ist die Verwendung von exit() nicht empfohlen.
Als Dokumentations-Programm soll Doxygen verwendet werden. Doxygen ist ein Open-Source-Dokumentationswerkzeug, um innerhalb von Quelltexten zu dokumentieren. Mit

dem Programm lassen sich alle auf C basierenden Pogrammiersprachen dokumentieren. Dabei ist anzumerken, dass Funktionen vor dem Prototypen und Programme auch Höhe des Präprozessors dokumentiert werden sollen. In der Dokumentation soll erklärt werden, was die Aufgabe der Funktion ist, welche Variablen verwendet werden und idealer Weise wie die Funktion funktioniert. Zum Beispiel müsste man angeben, welchen Algorithmus man verwendet.

3 Vorgehensweise

In den folgenden Textabschnitten werden wie in 1.1 beschrieben meine Vorgehensweise bei der Bearbeitung der Laboraufgaben dargestellt.

3.1 ReadMe aktualisieren

Die zweite Aufgabe ReadMe aktualisieren empfand ich als unkompliziert. Die in Teilaufgabe (a) zur Verfügung gestellten Beispiel Projekte 1 und 2 zum betrachten von ReadMe-Datei waren meiner Ansicht nach zum diesem Zeitpunkt etwas zu überkompliziert. Aus diesem Grund habe ich dazu entschieden das ReadMe des Veranstaltungs Repository als Vorlage zu verwenden. Nach diesem Vorbild habe ich die Teilaufgabe (b) ohne Schwierigkeiten erledigt. Für i. habe ich einen erläuternden Satz pro Verzeichnis geschrieben. Für ii. kann man den erforderlichen bash Befehl in der ReadMe Datei finden. Sowie eine Anleitung in Form einer Auflistung, um das Helloworld Projekt übersetzen und nutzen zu können.

3.1.1 Probleme in Aufgabe 2

Die einzigen Probleme mit dieser Aufgabe hatte ich mit der Teilaufgabe (c):
Die Anweisungen beziehungsweise Hilfestellungen in Verbindung mit der zu Verfügung gestellten Quelle hatten mich zu einem falschen Ansätz geführt. Durch experimentieren habe ich heraus gefunden, dass im Git-Lab Repository unter $gitlab.example.com/ < namespace > / < project > / - /settings/ci\_cd$ ein Link zufinden ist, der in die ReadMe-Datei einzufügen ist. Dadurch erscheint die Pipline-Badge auf der ReadMe-Datei. Dieser Link ist von Nutzer zu Nutzer unterschiedlich. Eine genaue Anleitung, wie dieser Link zufinden ist, ist in der Teilaufgabe 2c. zufinden.

3.2 Makefile mit Unit-Tests erstellen

Die Teilaufgaben 3a bis 3f stellten für mich kein Problem dar. Die vier Quelltexte *add.c*, *core.c*, *core.h* und *core_test.c* zu übernehmen und zu verbesseren sowie die makefile mit den geforderten Targets zu erstellen verlief ohne Schwiegkeiten. Die Fehler in den Quelltexten konnte ich mit Hilfsmittel identifizieren und beheben. Falls man bei diesem Schritt allerdings Schwierigkeiten hat empfiehlt es sich mittels:

```
gcc -Wall -Werror -c dateiname.c -odateiname.o
```

nach den Fehlern zu suchen.
Beim Erstellen der makefile Targets *all*, *clean*, *test*, *add*, *add.o*, *core.o*, *core_test.o* und *core_test* haben mir die 5.Vorlesung und die Seite GNU make geholfen. Da diese Targets standardmäSSig Teil vieler Projekte sind, waren für die Bearbeitung dieser Teilaufgaben keine weiteren Hilfsstellung nötig.

3.2.1 Ausführung Targets

Die Targets *all*, *add*, *add.o*, *core.o* erzeugen aus den Quelltexten Objektsdateien und aus diesen erzeugt das Target add eine für einen Nutzer ausführbare Datei. Die Erzeugung der Objektdatein wird mit Kompilierungsbefehlen ausgeführt. Beispielhaft mit *core.o* dargestellt.

```
core.o: core.c core.h
        gcc -Wall -Werror -c core.c -ocore.o
```

Ein Target zum erzeugen einer ausführbaren Datei hat die folgende Form:

```
add: add.o core.o
        gcc add.o core.o -oadd
```

Fügt man *add* zum .Phony Target *all* hinzu kann man mit einem Aufruf von all alle Objekt- und die ausführbare Datei erzeugen. weitere Hilfsstellungen finden Sie in den 2 Grundlagen Die Vorgängen und Abhängigkeiten gelten auch für die Erzeugung der Units-Tests, die über die Targets *core_test.o*, *core_test* und das .Phony-Target test aufgerufen werden.
Mit dem clean Target kann man nach den Verwendung des Projekts alle Objektdateien und ausführbare Dateien wieder löschen.

```
        rm ... -f
        /* mit dem Suffix -f kann man auch im Rahmen des Projekts erzeugte Verzeichnisse
        /* in den ... Bereich werden alle Dateien und Verzeichnisse geschrieben, die gel
```

3.2.2 argparse_int

Die Funktion argparse_int stellt eine benutzterfreundliche Alternative zu anderen Ganzzahlkonvertierungs-Funktionen zu Verfügung. Da diese Funktion Hauptbestandteil der weiterführenden Projekte dieses Labors ist, habe ich einen gro_ssen Wert daruaf gelegt diese vollständig und fehlerfrei zu implementieren. Die Funnktion benötigt 5 lokale Varibalen, drei Überprüfungen des übergebenen Strings sowie die eigentliche Konvertierung des Strings in einen Integer Wert.

```
int argparse_int(const char *str, int *value){
        char *end;
        char binary[] = {};
        char hexch = 'x';
        char b1 = '0';
```

```
char b2 = 'b';
/* looking the prefix */
/* if the str starts with a letter chancel*/
if(isalpha(str[0])){
        return 0;
}
/* if the str is a binary */
if(str[0] == b1 && str[1] == b2){
        for(int i=0; str[i]!='\0';i++){
                binary[i] = (char)str[i+2];
        }
        ...
/* hexadecimal */
else if(str[1] == hexch){
        ...
}
/* decimal */
else{
        ...
}
/* Konvertierung */
long int gzw = strtol(str, &end, 10);
value[0] = (int)gzw;
return 1;
```

Um diese Funktion ausreichend zu testen habe ich pro möglichen String Typen (hexadezimal, Binär, dezimal, positiv, negativ und Buchstabe) einen assert.Test in *core_test.c* implementiert. Idealer Weise sollte man allerdings nicht nur die erfolgreiche Ausführung der Funktion testen sondern auch das zu erwartende Ergebnis.
Hier ein kurzes Beispiel:

```
int intt;
const char t4[] = "0x0A";
        assert(1 == argparse_int(t4, &intt));
        assert(10 == intt);
```

3.2.3 coverage und Doxygen

Die Erzeugung der Coverage Analyse und der Dokumentation wird in der Makefile des jeweiligen Projekts ausgeführt. Ich hatte lange versucht mittels der 5 Vorlesung und Recherche im Internet herauszufinden wie die Syntax richtig zusetzen ist. Leider ohne Erfolg. Professor Bauer konnte mir in den Präzens Laborübungen zeigen wie man die Coverage-Analyse schreibt. Da ich diesen Vorgang bisher leider immernoch nicht verstanden habe zeige ich an dieser Stelle die verantwortlichen Code Ausschnitte.

```
core_test_coverage: core.c core_test.c core.h
        gcc --coverage -g core.c core_test.c -o core_test_coverage
...
.PHONY: coverage
coverage: core_test_coverage
        ./core_test_coverage
#       gcov ./core_test.c
#       gcov ./core.c
#       gcov ./add.c
        gcovr --html-details ./core_test_coverage
        firefox *.html
```

Mit diesen Befehlen werden mehrere Webseiten erstellt und geöffnet. Aus diesen kann man detaliert entnehmen, ob alle Funktionen getestet und auf Richtigkeit überprüft werden. Zur Umsetzung der Dokumentation benötigte ich keine weiteren Hilfstellungen außer die Anweisungen auf dem Laborblatt 3 Teilaufgabe 3k. Allerdings hatte ich einige Probleme bei der Installation der benötigten Packete. Die sind im Repositoryzufinden.
Mein Problem war, dass ich unter Debian root nicht alle Packete installieren konnte. Durch experimentieren mit der Konsole habe ich heraus gefunden, dass ich alle Packete einzeln installieren musste.
Unter root:

```
# apt install texlive-base
# apt install texlive-latex-extra
# apt install texlive-pictures
...
# apt install make
# apt install latexmk
...
```

Wenn man alle benötigten Packete installiert hat kann man ohne Schwierigkeiten die Anweisungen bis zur letzten letzten Teilaufgabe bearbeiten in der letzten Teilaufgabe muss ein Target zur automatischen Dokumentaionserzeugung geschrieben werden. Dabei gibt es einige Schwierigkeiten.
Erstens muss man die erzeugte PDF Datei aus dem Kinderverzeichnis latex in das Elternverzeichnis bewegen. Da dieser allerdings universal auf allen Geräten und Benutzern funktionieren muss, kann man nicht einfach den Pfad zum Verschieben angeben, weil dieser von Nutzer zu Nutzer unterschiedlich ist. Zweitens muss die PDF Datei dannach auch umbenannt werden. Professor Bauer hat mir den Tip gegeben das Linux Kommando mv zu verwenden. Daher habe ich mit den man pages zu mv und einer Internetseite ein Grundwissen für den Umgang mit mv angeeignet. Zudem habe ich während meiner Recherche auf der Seite Stackoverflow herausgefunden, wie man in einer makefile einen Verzeichniswechsel vornehmen kann.
Hier die vollständige Befehlsreihe:

```
doxygen: Doxyfile
        doxygen
        make -C latex
        cd latex; \
        mv refman.pdf ..
        mv refman.pdf add.pdf
```

3.3 Subtraktion

Da die Funktionen und Makefile Targets aus dem vorherigen Textabschnitt in allen weiteren Aufgaben Verwendung finden werde ich diese in den folgenden Textabschnitten nicht nocheinmal thematiesieren und nur neue Funktionen besprechen. Ein gutes Beispiel für diese Entscheidung ist das Subtraktions Projekt. Es ist nahezu identisch zu dem Add Projekt. Die einzigen Unterschiede sind Änderungen in den Namen und Testwerten sowie in die Hauptfunktion. In dieser wird allerdings ein Plus nur zu einem Minus ansonten sind die Funktion vollständig identisch.

```
int add(int x, int y)
{
        return x+y;
}
int sub(int x, int y){
        return x-y;
}
```

3.4 Größter Gemeinsamer Teiler

In dieser Aufgabe musste man eine Funktion schreiben, die aus zwei Integer Zahlen den größten gemeinsamen Teiler auf der Konsole ausgibt. Darüber hinaus sollte das Entprogramm zwei "$flags$" unterstützen $--verbose$ und $--lcm$. Mit $--verbose$ soll zusätzlich der Rechenweg ausgegeben werden und mit $--lcm$ soll der kleinste gemeinsame Teiler ausgegeben werden.
Ich bin bei der Implementierung der Flags so vorgegangen, dass ich alle dem Programm übergebenen Argumente nach den flags in Form von String durchsucht habe. Wurde eine flag gefunden wurden die verbleibenden Argumente in Integer-Werte umgewandelt und die geforderten Funktionen mit diesen durchgeführt. Beispiel anhand von $--lcm$:

```
        for(int i = 1; i < argc;i++){
                if(strcmp(argv[i],"--lcm") == 0 ){
                        if(i == 1){
                                printf("%i\n", scm(argv[2],argv[3]));
                                return 0;
                        }
                        else if(i == 2){
```

```
                        printf("%i\n", scm(argv[1],argv[3]));
                        return 0;
                }
                else{
                        printf("%i\n", scm(argv[1],argv[2]));
                        return 0;
                }
        }
}
```

Bei der Implementierung gcd(greatest common divisor) habe ich mich für den euklidischen Algorithmus entschieden, ebenso bei scm(smallest common multiple). Für die Implementierung von $--verbose$ habe ich das Prinzip von necho verwendet. Necho ist ein Programm, welches im Rahmen der "$GrundlagenderProgrammierung$" geschrieben werden sollte. Dabei werden die einzelnen Rechenschritte Zeile für Zeile auf der Konsole mittels der Verwendung von $printf$ ausgegeben. Funktionen für gcd:

```
int gcd(int a, int b){
        if(b == 0){
                return a;
        }
        else{
                return gcd(b, a % b);
        }

}

int verbose(int a, int b){
        int temp;
        while(1){
                if(b!=0){
                        printf("%i divided by %i = %i + %i\n", a, b, a/b, a%b);
                        temp = a;
                        a = b;
                        b = temp % a;
                }
                else if(b == 0){
                        printf("rest = 0 --> the gcd is %i\n", a);
                        return 1;
                }
        }
}

int scm(const char* str1, const char* str2){
```

```
        char *end;
        int max;
        int a;
        int b;
        a = strtol(str1, &end, 10);
        b = strtol(str2, &end, 10);
        if(strcmp(str1,str2) > 0){
                max = a;
        }
        else if(strcmp(str1,str2) < 0){
                max = b;
        }
        else{
                return a;
        }
        while(1){
                if((max % a == 0) && (max % b == 0)){
                        return max;
                }
                else{
                        max++;
                }
        }

}
```

3.5 Mysort

In dieser Aufgabe soll eine Funktion zum sortieren eines Integer Arrays geschrieben werden, da ich ein solches Programm schon einmal geschrieben habe konnte ich den Code aus diesem Programm fast eins zu eins übernehmen. Allerdings war dieses Programm in Java verfasst, daher musste leichte Anpassungen an C vornehmen. Auch Mysort sollte zwei flags unterstützen. Zum einen $--print-as-lines$ das heißt pro Zeile soll nur ein Element des Arrays ausgegeben werden und zum andereren $--reverse$. Wie der Name schon sagt soll das Array von hinten nach vorne auf der Konsole ausgegeben werden.
Für Mysort habe ich einen neuen und meiner Ansicht nach besseren Ansatz zur Überprüfung der flags verwendet. For-Schleifen bieten die Möglichkeit einen gegebenen Array schneller zu durchsuchen oder für Funktionen zu verwenden. Besonders wenn das Array größer wird ist die Verwendung von For-Schleifen von Vorteil. Ich habe $argc - 1$ als Länge eingesetzt. Dadurch kann man schnell den $Char - Array$ durchsuchen und in ein $Integer - Array$ ersetzen.

```
        int arrsort[argc-1];
        for(int i = 0; i <argc-1;i++){
                if(strcmp(argv[i],"--reverse")==0 || strcmp(argv[i],"--print-as-lines")==
```

```
                argparse_intArray(argv[i+2],i,arrsort);
        }
        argparse_intArray(argv[i+1],i,arrsort);
}
sort(arrsort,argc-1);
for(int j = 0; j <argc-1;j++){
        if(strcmp(argv[j],"--reverse")==0){
                reverse(arrsort,argc-2);
                return 0;
        }
        ....
}
```

Die drei Hauptfunktionen von Myort sind recht simple. Für $--print-as-lines$ habe ich bei der Ausgabe der Array-Elemente eine Zeilenumbruch pro Element hinzugefügt und bei $--reverse$ habe ich die Laufvaribale auf die Länge des Arrays gesetzt und diese pro Element dekrementiert bis der Anfang des Arrays erreicht ist. Als Sortierungsalgorithmus habe ich den Selectionsort-Algorithmus verwendet. Mittels von zwei For-Schleifen kann jedes einzelene Element nacheinander vergleichen. Sollte das aktuelle betrachtet Element größer als ein nachfolgendes Element im Array tauschen die beiden Elemente die Position im Array. Dieser Vorgang wird solange wiederholt, bis die äußere For-Schleife einmal das gesamte Array durchlaufen hat.

```
int sort(int arr[], int length){
        int temp[length];
        for(int i = 0; i < length-1;i++){
                for(int j = i + 1; j < length; j++){
                        if(arr[i] > arr[j]){
                                temp[i]=arr[i];
                                arr[i]=arr[j];
                                arr[j]=temp[i];
                                ...
}
```

4 Fazit

Abschließend kann ich sagen, dass durcgh die Bearbeitung der 3.Laborblattes ein besseres Verständis mit dem Umgang mit Linux und den Besonderheiten von C gewinnen konnte. Retrospektiv muss ich allerdings sagen, dass der uhrsprünglich geplante Zeitrahmen für die Bearbeitet der Aufgaben zu kurz war. Vor allem, weil einige wichtige Informationen bisher noch nicht besprochen wurden. Aufgrund dessen wurde die Frist auch angepasst. Trotzdem bin ich der Ansicht, dass die gegebene Zeit sehr knapp war. Gerade wenn man mit einbezieht, dass die Studierenden noch in anderen Module ähnlich schwere Aufgaben zu erledigen haben. Daher hätte ich mir gewünscht, dass das

dritte Labor etwas verkürtzt worden wäre, damit man die thematisierten Methoden und Grundlagen besser verinnerlichen könnte.

5 Literaturverzeichnis

Vorlesungen des Moduls

Programiz

Stack Exchange

GNU

Stack Overflow

Algorithms & Technologies