

# Betriebssysteme

## Verzeichnisse

Praktikum 7

Fachhochschule Bielefeld  
Campus Minden  
Studiengang Informatik

---

Beteiligte Personen:

Name	Matrikelnummer
Mirko Weidemann Kreitz	1048290
Oxana Zhurakovskaya	130157
Karsten Michael Tymann	1047529
Yuliia Dobranska	1093568

---

8. Juni 2016

## Inhaltsverzeichnis

1	Aufgabe 1 Typische Verzeichnisstrukturen	3
1.1	Fazit . . . . .	7
2	Aufgabe 2 Typische Verzeichnisstrukturen	8
2.1	Vorbereitung . . . . .	8
2.2	Durchführung . . . . .	8
2.3	Fazit . . . . .	19

## 1 Aufgabe 1 Typische Verzeichnisstrukturen

- Geben Sie die typische Verzeichnisstruktur einer aktuellen Linuxdistribution an. Was ist in den Verzeichnissen enthalten? Nennen Sie mindestens fünf Beispiele (wie `/etc`, `/usr` usw.). Finden Sie zusätzlich heraus, wo Log-Dateien gespeichert werden. Wo liegen ausgelagerte Inhalte des Hauptspeichers?

textbfAntwort: Ubuntu Verzeichnissystem entspricht Filesystem Hierarchy Standard(FHS): <sup>1</sup>

- `/` - Wurzelverzeichnis für alle anderen Linux Verzeichnisse. In der Regel entspricht dieses der Bootpartition (wenn nicht anders beim Installieren eingestellt), deswegen enthält es symbolische Verknüpfungen für `initrd.img` und `vmlinuz`, die im Ordner `/boot` liegen
- `/boot` - beinhaltet alle für den Systemstart(Booten) benötigten Dateien solche wie z.B Kernel <sup>2</sup>, initiale Ramdisk <sup>3</sup> und das Programm für den Memorytest `memtest86.bin`. Außerdem enthält dieser Verzeichnis Ordner `grub/` mit den Bootloader Dateien und dem Ordner `efi/` mit EFI-Programmen Dieses Verzeichnis muss beim Systemstart vorhanden sein.
- `/bin` - Enthält ausführbare Dateien (Programme). Es handelt sich bei den Dateien um System-Tools(z.B `cp`, `echo`, `mkdir`, `rm`), die von allen Benutzern genutzt werden( im Gegensatz zu `/sbin`). - Dieser Ordner darf keine weiteren Verzeichnisse enthalten. Dieses Verzeichnis muss beim Systemstart vorhanden sein.
- `/dev` - Dieses Verzeichnis beinhaltet alle für den Zugriff auf die Geräte erforderliche Dateien (z.B. für Festplatten, DVD-Laufwerke, Maus, Monitor). Hier werden z.B. Festplatten Partitionen eingebunden. Die Dateien werden als Schnittstellen für Hardware benutzt. (ausgenommen sind jene die mit hot-Plugin eingebunden werden, dafür gibt es den Ordner `/udev`) Diese Verzeichnis muss beim Systemstart vorhanden sein.
- `/etc` - steht für “*editable text configuration*”. Hier liegen systemweit gültige Konfigurations- und Informationsdateien des Basissystems. Hier findet man solche Dateien wie z.B. `fstab`, `hosts`, `lsb-release`, `blkid.tab`. Diese Konfigurationsdateien können von gleichnamigen Dateien im Homeverzeichnis überschrieben werden. Der Ordner enthält viele Verzeichnisse mit Konfigurationsdateien. Beispiele:
  - \* `/etc/opt`: Verzeichnisse und Konfigurationsdateien für Programme in `/opt`
  - \* `/etc/network`: Verzeichnisse und Konfigurationsdateien des Netzwerkes (Interfaces u.s.w.)
  - \* `/etc/init.d`: Enthält Start- und Stopskripte
  - \* `/etc/hosts` Einstellungen für IP-Adressen Auflösung

<sup>1</sup> Quellen: [www.pcwelt.de](http://www.pcwelt.de), [www.selflinux.org](http://www.selflinux.org), [jankarres.de](http://jankarres.de), [wiki.ubuntuusers.de](http://wiki.ubuntuusers.de),

<sup>2</sup> für Desktop `vmlinuz-versionsnummer-generic`, für Server `vmlinuz-versionsnummer-server`, für virtuelle Maschinen `vmlinuz-versionsnummer-virtual`

<sup>3</sup> `initrd.img-versionsnummer-generic/-server/-virtual`

- \* */etc/ssh/* enthält SSH Konfigurationsdateien
- \* */etc/X11* Konfigurationen für grafische X-Window-Subsystem

Dieses Verzeichnis muss beim Systemstart vorhanden sein.

- **/home** - Dieses Verzeichnis enthält ein Home-Ordner jedes Benutzers (als Name wird der Benutzername übernommen). Hier speichern die Benutzer Persönliche Daten, Eigene Programme, und Konfigurationsdateien die nur für den konkreten Benutzer beim Einloggen verfügbar sind. Der User hat volle Schreib- und Leserechte für das eigene Homeverzeichnis. Dieser Ordner kann auf eine andere Partition verlegt werden. Dieses Verzeichnis muss nicht beim Systemstart vorhanden sein.
- **/lib, /lib32, /lib64** - hier liegen wichtige dynamische Bibliotheken des Systems und Kernel-Modules, viele werden beim Systemstart benötigt. z.B:
  - \* */lib/modules* - Kernelmodule
  - \* */lib/udev*: Bibliotheken und Programme für udev
  - \* */lib/linux-restricted-modules*: Speicherort für eingeschränkte Treiber (z.B. Grafikkarte)
- **/lost+found** - Das Verzeichnis ist in ext2, ext3 und ext4 Dateisystemen vorhanden. Der Ordner beinhaltet Dateien auf der Festplatte, die in der Verzeichnissstruktur nicht mehr zugeordnet werden können (bei System-/Programmabstürze oder Hardware-Fehler). Bei einem normal funktionierenden System soll der Ordner leer bleiben.
- **/media** - Das Verzeichnis wird nur als Einhängpunkt für Wechseldatenträger (solche wie Disketten */media/floppy*, CD-/DVD-Laufwerke */media/cdrom*, */media/dvd*, Zip-Disks *media/zip*, externe USB-Festplatten). Das Verzeichnis muss auf der gleichen Partition mit dem Root-Verzeichnis liegen.
- **/mnt** - Der Verzeichnis wird als Einhängpunkt für temporär verfügbare Dateisysteme (z.B. eine Windows Partition um auf die dort gespeicherten Daten zuzugreifen). Theoretisch könnte man das Verzeichnis als Einhängpunkt für Wechseldatenträger nutzen, dies widerspricht aber den FHS Standard.
- **/opt** - Dieser Ordner wird für die manuelle Installation von Programmen genutzt. Das Verzeichnis ist optional und ist nicht für den Start des Systems notwendig. Deswegen kann er auch auf andere Partition verlagert werden.
- **/proc** - Das Verzeichnis beinhaltet Prozess- und Systeminformationen und stellt die Schnittstelle zum Kernel dar und ist im eigentlichen Sinne ein spezielles, virtuelles Dateisystem. z.B: Beispiele: *version* - Kernelversion, *swaps* Swapspeicherinformationen, *cpuinfo*, *interrupts*, usw. Außerdem liegen hier Verzeichnisse mit Prozessnummer als Name, die Dateien zu Prozessinformationen enthalten, solche wie *status* Prozessstatus, *limits* Limits für Ressourcen Verbrauch. Dieser Ordner ist nicht FHS Standard aber standardmäßig in Linux Distribution vorhanden. Muss beim Systemstart vorhanden sein.

- **/root** - Das ist das Homeverzeichnis des Superusers (root). Hier hat nur der Superuser volle Schreib- und Leserechte. Dieser Ordner muss immer vorhanden sein.
  - **/run** - Der Ordner wird für Dateien von laufenden Prozessen genutzt. Hier liegen die meisten PID-Files *\*.pid* (Process identifier).
  - **/sbin** - Hier liegen system binaries, die für essentielle Aufgaben der Systemverwaltung notwendig sind. Diese Programme können nur vom Superuser ausgeführt werden. Der Ordner muss auf der Rootpartition immer vorhanden sein.
  - **/srv** - Hier liegen Dateien für System-Dienste. Oft werden hier variable Dateien solche wie Logfiles oder Mails eines Webserver gespeichert, (in unserem Ubuntu 14.04 ist dieser Ordner leer). Das Verzeichnis ist optional und ist nicht in allen Linuxdistributionen vorhanden.
  - **/sys** - Das Verzeichnis hat eine ähnliche Funktion wie */proc* und beinhaltet hauptsächlich Kernelschnittstellen. Der Ordner ist im FHS noch nicht spezifiziert.
  - **/tmp** - Wird für temporäre Dateien von Programmen genutzt. Nach FHS standart wird dieser Ordner nach jedem Neustart bereinigt. Jeder Benutzer hat vollen Zugriff nur auf seine eigenen temporären Dateien.
  - **/usr** - Diese Verzeichnisstruktur enthält die meisten Systemtools, Bibliotheken und installierten Programme. Hier werden Programme gespeichert die von Paketverwaltung installiert wurden. Es liegen folgende Unterordner vor:
    - \* */usr/bin* - Hier werden Ausführbare Programme gespeichert
    - \* */usr/include* - Hier befinden sich Header-Dateien, die man bei der C-Programmierung im Source-Code inkludiert.
    - \* */usr/lib* - Weitere Programm-Bibliotheken
    - \* */usr/local* - Der Ordner hat die gleiche Struktur wie */usr* selbst, und kann für manuell installierte Programme genutzt werden (wie */opt*).
    - \* */usr/sbin* - Hier liegen optionale Systemprogramme
    - \* */usr/share* - Hier liegen sich nicht ändernde, architekturunabhängige Dateien.
    - \* */usr/share/applications* - Hier findet man Programmstarter, die für Anwendungsmenüs genutzt werden
    - \* */usr/share/man* - Beinhaltet Man-pages
  - **/var** - Wird zur Speicherung von veränderlichen Daten genutzt. Hier liegen nur Verzeichnisse, deren Inhalt regelmäßig verändert wird. z.B. in */var/log* In dem Ordner werden ständig Logdateien überschrieben und neu angelegt.
- Geben Sie die typische Ordnerstruktur von Microsoft Windows an. Nennen Sie analog zur vorherigen Aufgabe einige beispielhafte Inhalte der jeweiligen Verzeichnisse (wie

z.B, *C:\Windows\System32*). **Antwort:** Windows Ordner: (Pfade/Bezeichnungen können je nach Betriebssystem abweichen) <sup>4</sup>

- **System32:** *C:\Windows\System32* - ist das Systemverzeichnis von Windows. Hier liegen für das System essentielle Daten, Programme und Bibliotheken

Inhalte:

- \* *C:\Windows\System32\Configuration* - Konfigurationsdateien.
- \* *C:\Windows\System32\config* - Registry Dateien und Event logs.
- \* *C:\Windows\System32\de-DE* - hier liegen die Lokalisierungspakete für die deutsche Sprache
- \* *C:\Windows\System32\GroupPolicy* - administrative Templates und Script-files (Gruppenrechte)
- \* *C:\Windows\System32\Microsoft* - Kryptographie Dateien (Verschlüsselung)

- **Program Files:** *C:\Program Files(x86)* - ist der standard Ordner von Microsoft indem Anwendungen gespeichert werden, die nicht zum Betriebssystem gehören. Jede Anwendung erhält zusätzlich ein Unterverzeichnis für ihre Ressourcen. Die Bezeichnung kann je nach Betriebssystem den Zusatz (x86) bzw. (x64) mit sich führen. Im Ordner *Program Files(x86)* werden 32 Bit Anwendungen und im Ordner *Program Files(x64)* 64 Bit Anwendungen standardmäßig gespeichert.

Inhalte:

- \* *C:\Program Files\Adobe*
- \* *C:\Program Files\MSBuild*
- \* *C:\Program Files\Microsoft.NET*
- \* *C:\Program Files\Mozilla Firefox*
- \* *C:\Program Files\Windows Media Player*

- **Desktop:** *C:\Users\Username\Desktop* - existiert zum Einen als Sonderverzeichnis und zum Anderen als virtuelles Verzeichnis. Das Sonderverzeichnis enthält die Dateien, die auf dem Desktop des Benutzers gespeichert sind. Beim virtuellen Verzeichnis handelt es sich um den Windows-Desktop. Ein Sonderverzeichnis hat einen Bezug zu einem echten Verzeichnis auf dem Dateisystem.

Inhalte:

- \* *C:\Username\Desktop\4.Semester*
- \* *C:\Username\Desktop\Urlaubsfotos*
- \* *C:\Username\Desktop\BSPraktikum*
- \* *C:\Username\Desktop*
- \* *C:\Username\Desktop\Softwareprojekt*

---

<sup>4</sup> Quellen: [www.technet.microsoft.com](http://www.technet.microsoft.com),  
[www.wikipedia.org/wiki/Program\\_Files](http://www.wikipedia.org/wiki/Program_Files),

[www.wikipedia.org/wiki/Sonderverzeichnis](http://www.wikipedia.org/wiki/Sonderverzeichnis),

- **Favoriten:** *C:\Users\sername\Favorites* - Hier befinden sich die Favoriten des Benutzers. Links die im Browser als Favoriten hinterlegt sind. Favorisierte Dokumente und Verzeichnisse des Nutzers werden im Ordner *Links* gespeichert.  
*C:\Users\Username\Links*

Inhalte:

- \* *C:\Users\Username\Favorites\Acer\Acer.url*
- \* *C:\Users\Username\Favorites\Acer\eBay.url*

- **Eigene Dateien:** *C:\Users\Username* - Hier befinden sich die Dokumente/-Verzeichnisse des Benutzers.

Inhalte:

- \* *C:\Users\Username\Pictures*
- \* *C:\Users\Username\Music*
- \* *C:\Users\Username\Videos*
- \* *C:\Users\Username\Favorites*
- \* *C:\Users\Username\Links*

## 1.1 Fazit

In der Linux Distribution werden Log Dateien nach FHS Standart im Verzeichnis */var/log* hinterlegt. In Linux werden Inhalte des Hauptspeichers auf eine spezielle swap-Partition ausgelagert. Wir können in */proc/swap* Dateien sehen, die in unserem konkreten Ubuntu-System die Inhalte auf */dev/sda4* auslagert.

Bei Windows und Linux gibt es Verschiedene Standarte für Verzeichnis Struktur.

## 2 Aufgabe 2 Typische Verzeichnisstrukturen

Bearbeiten Sie die folgenden Aufgaben und protokollieren Sie Ihr Vorgehen mithilfe der Vorlage. Entwickeln Sie ein Programm `myls`, das den Inhalt von Verzeichnissen ausgibt. Die grundlegende Funktion ist in etwa vergleichbar mit dem Shell-Kommando `ls`.

### 2.1 Vorbereitung

Zur Testzwecken haben wir einen Test Ordner erstellt *testDir*. im Ordner haben wir einige Dateien und Verzeichnisse angelegt(darunter ein symlink und versteckte Dateien):

```
cd testDir
touch fileOwnerUserGroupUser
touch fileOwnerUserGroupTest
touch file1
touch .hiddenFile
touch file.c
touch file.cpp
touch file2.c
mkdir dir1
mkdir .dirHidden
touch ausfuerbar
ln -s fileOwnerUserGroupTest symLinkToFileOwnerUserGroupTest
```

wir legen auch eine test Gruppe an, um das Ergebnis der `myls` mit Parametern `-g` und `-o` sichtbar und besser interpretierbarer zu machen, denn von uns erstellte Dateien haben als default `userid = 1000` und `Gruppenid = 1000`

```
sudo addgroup test
Legen Gruppe test (GID 1001) an ...
Fertig.
```

nun haben wir eine Gruppe mit Id 1001

wir ändern die Gruppe von File `fileOwnerUserGroupTest` zu Gruppe mit Id 1001

```
sudo chown -c :test fileOwnerUserGroupTest
```

File *ausfuerbar* wird ausführbar gemacht

### 2.2 Durchführung

- Zuerst definieren wir die maximale Länge für Pfad `#define MAX_PATH 1024`
- Wir definieren eine Variable die den aktuellen Pfad speichert `char path[MAX_PATH];`
- Der Name des auszulesenden Verzeichnisses soll dem Programm als Argument übergeben werden. Wird kein Verzeichnis angegeben, so wird das lokale Verzeichnis ausgegeben.
- Hierzu nutzen wir die Eingabeparameter der `main`-Methode.  
`main(int argc, char *argv[])` Um Parameter annehmen zu können nutzen wir



die Variable `argc` um die Anzahl der übergebenen Argumente zu zählen sowie das Array `argv` um die Argumente auslesen zu können.

- Im nächsten Schritt prüfen wir die Anzahl der übergebenen Argumente.  
`if (argc > 1 && strncmp(argv[1], "-", 1) != 0)`  
Wurden Argumente übergeben so ist die Variable `argc` größer als 1. Zusätzlich prüfen wir ob das erste übergebene Argument mit einem - anfängt (mittels `strncmp` aus header `string.h`, es wird nur das erste Zeichen geprüft). Dies würde bedeuten dass der User das `listing` aus dem lokalen Pfad ausführen möchte da das - Zeichen das Zeichen für die Parameter ist. Ist dies der Fall so wird der `else`-Teil ausgeführt. Dieser wird auch ausgeführt wenn keine Argumente übergeben werden. `getcwd(path, MAX_PATH);` Im `else`-Teil wird dann das aktuelle Working Directory mittels `getcwd` ermittelt und dem Array `path` übergeben. Wurde ein Pfad übergeben so betrachten wir den `If` Block. `strcpy(path, argv[1]);` Hier überweisen wir dem `path` Array den vom User übergebenen Pfad.
- Danach bauen wir eine Abfrage ein ob ein übergebener Pfad mit einem / endet. Ist dies nicht der Fall so hängen wir eines an. Hierzu ermitteln wir zunächst die Länge des Pfades: `int len = strlen(path);` Dann erzeugen wir einen Pointer der auf den letzten Index des Arrays zeigt `const char *last = &path[len - 1];` Nun prüfen wir ob dieses Zeichen ein / ist `if (strcmp(last, '/') != 0)` Ist dies nicht der Fall so hängen wir eins an. `strcat(path, "/");` Diese Methoden der String-Manipulation entstammen dem `string.h` Header.
- Wir definieren eine Variable `int c;` wo wir eine einzige Option aus der übergebenen Options Liste speichern
- Um die Optionen für `mys` zu speichern und weiter auszuwerten legen wir drei Variablen an
  - `int aoption = 0` für Option `-a` (gültige Werte 0 oder 1)
  - `int loption = 0;` für `-l` und `g` Option (gültige Werte 0, 1, 2)
  - `int ooption = 0;` für `-o` Option (gültige Werte 0 oder 1)
- Wir nutzen die Funktion `getopt(argc, argv, 'algo')` aus dem Header `unistd.h`, die das Auslesen der Optionen aus dem Argument-string erleichtert. Als ersten und zweiten Argument werden die Argumente die die `main` Methode erhält an `getopt()` übergeben, das dritte Argument übergeben wir als String aus allen gültigen Argumenten, nach denen durchgesucht wird. Wenn es keine Argumente in `argv0` gibt, dann liefert die Methode den Wert `-1` zurück.
- Den Rückgabewert von `getopt` nutzen wir als Abbruch Bedingung für die `while`-Schleife  
`while ((c = getopt(argc, argv, 'algo')) != -1)`
- In der `while`-Schleife prüfen wir mit `switch-case` welche Option ausgelesen wurde, und ändern die Werte von den oben genannten Variablen: `a`-,`l`-,`ooption`. Wird beispielsweise ein `a` gelesen so wird der zutreffende case ausgeführt und das `aoption`

Flag auf 1 gesetzt. Bei einem gelesenen l wird die loption auf 1 gesetzt. Wird ein g gesetzt so wird die loption auf 2 gesetzt, folglich erkennen wir dass das längere Ausgabeformat aber ohne UserID gewünscht ist. Wird ein o gesetzt so setzen wir die ooption auf 1. Zudem prüfen wir ob das loption Flag bereits gesetzt wurde. Wurde beispielsweise nur ein o als Parameter übergeben so müssen wir das loption Flag auf 1 setzen damit wir wissen dass das längere Ausgabeformat gewünscht ist. Die ooption sorgt dafür dass die GroupID ausgeblendet werden soll. Kombinationen der Parameter sind somit möglich.

- Danach rufen wir die Funktion `readPath(path, aoption, loption, ooption);` auf. Wir übergeben hier den Pfad sowie die 3 Flags.

```
int main(int argc, char *argv[]) {
    if (argc > 1 && strncmp(argv[1], "-", 1) != 0) {
        strcpy(path, argv[1]);
    } else {
        getcwd(path, MAX_PATH);
    }
    int len = strlen(path);
    const char *last = &path[len - 1];
    if (strcmp(last, "/") != 0) {
        strcat(path, "/");
    }
    printf("input path: %s", path);
    int c;
    int aoption = 0;
    int loption = 0;
    int ooption = 0;
    while ((c = getopt(argc, argv, "algo")) != -1) {
        switch (c) {
            case 'a':
                aoption = 1;
                break;
            case 'l':
                loption = 1;
                break;
            case 'g':
                loption = 2;
                break;
            case 'o':
                if (loption == 0)
                    loption = 1;
                ooption = 1;
                break;
        }
    }
}
```

```

        readPath(path, aoption, loption, ooption);
        return 0;
    }

```

- Um den Verzeichnis Inhalt und Informationen auszulesen definieren wir eine Funktion `void *readPath(char *path, int aoption, int loption, int ooption)`
- Die Funktion nimmt als erstes Argument den Verzeichnisnamen, deren Inhalt ausgelesen wird, die 3 folgenden Argumente sind die FLags.
- Um Ordner Information zu lesen nutzen wir die Bibliothek `dirent.h`,
- Wir legen uns eine Variable für den aufgelösten Pfad an, mit der bereits definierten Größe des maximalen Pfades `char resolved_path[MAX_PATH]`; Außerdem definieren wir eine Variable `DIR *dir = NULL`;, die den Directory Stream beinhalten wird.

Um den Ordner Inhalt aus dem directory stream zu lesen, definieren wir eine Variable `struct dirent *dptr = NULL`;

Und eine Char Variable die lediglich einen Punkt hält, die wir zum Vergleich nutzen werden. `char *dot = '.'`;

- Um den absoluten Pfad zu erhalten nutzen wir die Funktion `realpath((char*)path, resolved_path)` aus der Standardbibliothek, als Parameter übergeben wir den vom Benutzer übergebenen Pfad `path` und unsere Variable `resolved_path`, die das Ergebnis erhalten wird, dabei casten wir die Variable `path` zu einem char pointer. Die Funktion wird einen NULL pointer zurückliefern, falls beim Pfadname auflösen ein Fehler auftritt, deswegen können wir das Ergebnis der Variable `realpath` in der If-Abfrage überprüfen, und falls etwas mit dem Pfad nicht stimmt, kann die Funktion `readPath` ihre Arbeit abbrechen.
- Wenn der Pfad erfolgreich aufgelöst wurde können wir weiter vorgehen.
- Wir öffnen den directory stream mit der Funktion `opendir(resolved_path)`, die Funktion wird einen NULL Pointer zurückliefern wenn ein Fehler beim Öffnen auftritt. Wir fragen das Resultat ebenfalls in der If-Abfrage ab, wenn der Ordner erfolgreich geöffnet wurde, kann man weiter vorgehen, ansonsten muss die Funktion `readPath` ihr Arbeit abbrechen.

```
if ((dir = opendir(resolved_path)))
```

- Nun kann man mit der While-Schleife durch die einzelnen Einträge im directory stream iterieren, dabei hilft uns die Variable `dptr`, die bei jeder Iteration auf den nächsten Eintrag zeigt. Um den nächsten Eintrag auszulesen, benutzen wir die Funktion `readdir(dir)`, die als Parameter einen directory stream annimmt
- ```
while ((dptr = readdir(dir)))
```

- Innerhalb der while-Schleife prüfen wir ob das aoption Flag NICHT gesetzt ist `if (!aoption)`

Wir schließen die Dateien die mit dem Namen `.` beginnen aus, indem wir den Namen des aktuellen Eintrags mit der Variable `dot` vergleichen. Und wenn es sich um diese Dateien handelt, geht die while schleife ohne weiteres Vorgehen zur nächsten Iteration.

```
if (strncmp(dptr->d_name, dot, 1) == 0) continue;
```

- Ist das a-Flag also gesetzt so nehmen wir auch die Versteckten Dateien mit.
- Nun prüfen wir ob das l-Flag gesetzt wurde `if (loption!=0)`  
Ist dies der Fall, so legen wir uns eine Struktur vom Typ `stat` an, wo wir Dateiinformationen speichern können. `struct stat lstruct;`  
Die Methode `printlstruct(dptr->d_name,loption, ooption);` wird später erklärt. Sie erhält als Eingabeparameter den aktuellen File-Namen, sowie die l- und o-Flags.

- Mittels `lstat(dptr->d_name, &lstruct)` lesen wir die Information von der aktuellen Datei aus und Speichern diese in der vorher definierten `lstruct` Variable. Dabei folgt die Funktion im Gegensatz zur Funktion `stat()` nicht dem symbolic link, sondern es wird die Information über den Link selbst ausgelesen und nicht über die referenzierte Datei. Wenn die Information erfolgreich ausgelesen wurde, liefert die funktion den Wert 0 zurück, sonst den Wert -1
- Wir nutzen den Rückgabewert von der Funktion `lstat` in der if abfrage, um zu prüfen ob der `lstat` Aufruf funktioniert hat.
- Nun wird geprüft, ob es sich bei der Datei um eine ausführbare Datei handelt. Ist dies der Fall so soll die Datei rot ausgegeben werden.
- Dazu prüfen wir innerhalb der if-Abfrage ob der User, die Gruppe oder oder Andere execute-Rechte haben. Zusätzlich prüfen wir ob es eine Ausführbare Datei ist `S_IXEXEC`. Dieses Attribut ist allerdings bereits durch `S_IXUSR` abgelöst worden.

Um also auf diese Rechte vergleichen zu können lesen wir aus dem struct den `mode_t` aus in denen diese Flags gesetzt sind. Der Vergleich ob das Flag gesetzt ist erfolgt über den logischen UND-Operator. (`lstruct.st_mode & S_IXUSR`)

Diese Abfrage zieht sich für die anderen Abfragen so durch.

```
if((lstat(dptr->d_name, &lstruct) == 0 &&
((lstruct.st_mode & S_IXUSR) ||
(lstruct.st_mode & S_IXGRP) ||
(lstruct.st_mode & S_IXOTH) ||
(lstruct.st_mode & S_IXEXEC) ))
```

- Ist also der Aufruf von `lstat` geglückt, und ist die Datei eine ausführbare Datei, so legen wir den Farb Code für die nächste Ausgabe fest.

```
printf("\033[0;31;1m");
```

Die Eröffnung der Sequenz ist dabei die `\033[`.

Es folgt die Hintergrundfarbe die wir gerne bei schwarz belassen `0;`.

Nun die Schriftfarbe `31;`. Der Farbcode entspricht Rot.

Es folgt die Vordergrundfarbe `1m` die keinen Effekt hat. Das `m` schließt die Sequenz ab. Nun ist jeder folgende Output Rot.

- Um die Dateien mit Endung `.c` herauszufiltern, und diese anschließend grün zu färben ermitteln wir zuerst die Länge des aktuellen Dateinamens und speichern diese in einer Variable

```
int len = strlen(dptr->d_name);
```

und wir erzeugen einen Pointer zum vorletzten Zeichen im Dateinamen

```
const char *last_two = &dptr->d_name[len - 2];
```

Anschließend prüfen wir mit der Funktion `strcmp(last_two, “.c”)` anhand der letzten zwei Zeichen ob es sich um die Endung `“c”` handelt. Wenn dies der Fall wird die Ausgabe grün gefärbt `printf("\033[0;32;1m");`

- Außerhalb des `if`-Blocks wird dann der Name der aktuellen Datei ausgegeben `printf(“%sn”, dptr->d_name);`.
- Falls der `loption` Flag gesetzt war, so muss nun die Textfarbe wieder zurückgesetzt werden `if (loption!=0) printf("\033[0;0;0m");`.
- Ist die `while`-Schleife durchgelaufen so schließen wir den `directory stream` `closedir(dir);`.

```
void *readPath(char *path, int aoption, int loption, int ooption) {
char resolved_path[MAX_PATH];
DIR *dir = NULL;
struct dirent *dptr = NULL;
char *dot = “.”;
if (realpath(path, resolved_path)) {
printf("resolved_path: %s\n", resolved_path);
if ((dir = opendir(resolved_path))) {
while ((dptr = readdir(dir))) {
if (!aoption) {
if (strncmp(dptr->d_name, dot, 1) == 0) {
continue;
}
}
if (loption!=0) {
struct stat lstruct;

printf("%s", dptr->d_name, loption, ooption);

if (lstat(dptr->d_name, &lstruct) == 0 &&
```

```

        ((lstruct.st_mode & S_IXUSR) ||
         (lstruct.st_mode & S_IXGRP) ||

         (lstruct.st_mode & S_IXOTH) ||
         (lstruct.st_mode & S_IXEXEC) )){
            printf("\033[0;31;1m");
        }

        int len = strlen(dptr->d_name);
        const char *last_two = &dptr->d_name[len - 2];
        if (strcmp(last_two, '.c') == 0) {
            printf("\033[0;32;1m");
        }
    }
    printf("\%s\n", dptr->d_name);
    if (loption!=0)
        printf("\033[0;0;0m");
    }
    closedir(dir);
}
}
return NULL;
}

```

- Um die ausführlichen Informationen über den File auszugeben, definieren wir eine Funktion

```
printlstruct(char * filename, int loption, int ooption)
```

- Um auf die Fileattribute zugreifen zu können definieren wir die Struktur  
`struct stat lstruct;`
- Wir brauchen auch einen vollen Pfad für die Dateinamen für die Funktion `lstat`  
Dafür definieren wir eine Variable `char fullpath[MAX_PATH];` und dann bauen wir den aus der in der Variable `path` vorhandenen Pfad zum Verzeichnis und aus dem aktuellen Dateinamen zusammen.

```
strcpy(fullpath, path);
strcat(fullpath, filename);
```

- Diese Pfad übergeben wir zusammen mit `lstruct` an die Funktion `lstat(fullpath,&lstruct);`
- Anschließend überprüfen wir ob die Lese-Schreib-Execute Rechte für den Owner, die Gruppe und Andere gesetzt sind. Hierzu prüfen wir die einzelnen Flags und geben dann bei Erfolg oder Misserfolg das entsprechend Zeichen aus.

```

printf( (lstruct.st_mode & S_IRUSR) ? "r" : "-");
printf( (lstruct.st_mode & S_IWUSR) ? "w" : "-");
printf( (lstruct.st_mode & S_IXUSR) ? "x" : "-");
printf( (lstruct.st_mode & S_IRGRP) ? "r" : "-");
printf( (lstruct.st_mode & S_IWGRP) ? "w" : "-");
printf( (lstruct.st_mode & S_IXGRP) ? "x" : "-");
printf( (lstruct.st_mode & S_IROTH) ? "r" : "-");
printf( (lstruct.st_mode & S_IWOTH) ? "w" : "-");
printf( (lstruct.st_mode & S_IXOTH) ? "x\t" : "-\t");

```

- Es folgt die Ausgabe der verschiedenen Strukturelemente der stat Struktur.
- Nun geben wir die Anzahl der Links auf die Datei aus.  

```
printf("%ld", (long) lstruct.st_nlink);
```
- Für die UserID des Dateibesitzers prüfen wir ob das g Flag nicht gesetzt wurde. Dann soll es ausgeführt werden.  

```
if(loption!=2) printf("\t%ld", (long) lstruct.st_uid);
```
- Nun prüfen wir ob das o Flag gesetzt wurde. Ist dies nicht der Fall so geben wir die GroupID des Dateibesitzers aus.  

```
if(ooption==0) printf("\t%ld", (long) lstruct.st_gid);
```
- Es folgt die Ausgabe der Dateigröße in Bytes  

```
printf("\t%lld", (long long) lstruct.st_size);
```
- Als nächstes folgen die Ausgaben für die Zeitpunkte des letzten Zugriffs, der letzten Modifikation und der letzten Statusänderung.  

```

printf("\t%s", ctime(&lstruct.st_atime));
printf("\t%s", ctime(&lstruct.st_mtime));
printf("\t%s", ctime(&lstruct.st_ctime));

```
- Abschließend die Ausgabe der I/O Block Größe.  

```
printf("\t%ld ", (long) lstruct.st_blksize);
```

```

void printlstruct(char * filename, int loption, int ooption){
    struct stat lstruct;
    char fullpath[MAX_PATH];
    strcpy(fullpath, path);

    strcat(fullpath, filename);
    lstat(fullpath, &lstruct);

```

```

printf( (lstruct.st_mode & S_IRUSR) ? "r" : "-");
printf( (lstruct.st_mode & S_IWUSR) ? "w" : "-");
printf( (lstruct.st_mode & S_IXUSR) ? "x" : "-");
printf( (lstruct.st_mode & S_IRGRP) ? "r" : "-");
printf( (lstruct.st_mode & S_IWGRP) ? "w" : "-");
printf( (lstruct.st_mode & S_IXGRP) ? "x" : "-");
printf( (lstruct.st_mode & S_IROTH) ? "r" : "-");
printf( (lstruct.st_mode & S_IWOTH) ? "w" : "-");
printf( (lstruct.st_mode & S_IXOTH) ? "x\t" : "-\t");

printf("%ld ",(long) lstruct.st_nlink);
if(loption!=2)
    printf("\t%ld ",(long) lstruct.st_uid);
if(ooption==0)
    printf("\t%ld ",(long) lstruct.st_gid);
printf("\t%lld ",(long long) lstruct.st_size);
printf("\t%s ",ctime(&lstruct.st_atime));
printf("\t%s ", ctime(&lstruct.st_mtime));
printf("\t%s ", ctime(&lstruct.st_ctime));
printf("\t%ld ",(long) lstruct.st_blksize);
}

```

Gesamte Code:

```

/*
 * myls.c
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define MAX_PATH 1024

char path[MAX_PATH];

void printlstruct(char * filename, int, int);
void *readPath(char *path, int, int, int);

int main(int argc, char *argv[]) {

```



```

    if (argc > 1 && strcmp(argv[1], "-") != 0) {
        strcpy(path, argv[1]);
    } else {
        getcwd(path, MAX_PATH);
    }
    int len = strlen(path);
    const char *last = &path[len - 1];
    if (strcmp(last, "/") != 0) {
        strcat(path, "/");
    }

    printf("input path: %s", path);
    int c;
    int aoption = 0;
    int loption = 0;
    int ooption = 0;
    while ((c = getopt(argc, argv, "algo")) != -1) {
        switch (c) {
            case 'a':
                aoption = 1;
                break;
            case 'l':
                loption = 1;
                break;
            case 'g':
                loption = 2;
                break;
            case 'o':
                if (loption == 0)
                    loption = 1;
                ooption = 1;
                break;
        }
    }

    readPath(path, aoption, loption, ooption);
    return 0;
}

void *readPath(char *path, int aoption, int loption, int ooption) {
    char resolved_path[MAX_PATH];
    DIR *dir = NULL;
    struct dirent *dptr = NULL;
    char *dot = ".";
    if (realpath(path, resolved_path)) {

```

```

printf("resolved_path: %s\n", resolved_path);
if ((dir = opendir(resolved_path))) {
while ((dptr = readdir(dir))) {
if (!aoption) {
if (strcmp(dptr->d_name, dot, 1) == 0) {
continue;
}
}
if (loption!=0) {
struct stat lstruct;

printlstruct(dptr->d_name, loption, ooption);

if(lstat(dptr->d_name, &lstruct) == 0 &&
((lstruct.st_mode & S_IXUSR) ||
(lstruct.st_mode & S_IXGRP) ||
(lstruct.st_mode & S_IXOTH) ||
(lstruct.st_mode & S_IEXEC) )){
printf("\033[0;31;1m");
}

int len = strlen(dptr->d_name);
const char *last_two = &dptr->d_name[len - 2];
if (strcmp(last_two, ".c") == 0) {
printf("\033[0;32;1m");
}

}
printf("%s\n", dptr->d_name);
if (loption!=0)
printf("\033[0;0;0m");

}
closedir(dir);
}
return NULL;

}

void printlstruct(char * filename, int loption, int ooption){
struct stat lstruct;
char fullpath[MAX_PATH];
strcpy(fullpath, path);

strcat(fullpath, filename);

```

```

lstat(fullpath,&lstruct);

printf( (lstruct.st_mode & S_IRUSR) ? "r" : "-");
printf( (lstruct.st_mode & S_IWUSR) ? "w" : "-");
printf( (lstruct.st_mode & S_IXUSR) ? "x" : "-");
printf( (lstruct.st_mode & S_IRGRP) ? "r" : "-");
printf( (lstruct.st_mode & S_IWGRP) ? "w" : "-");
printf( (lstruct.st_mode & S_IXGRP) ? "x" : "-");
printf( (lstruct.st_mode & S_IROTH) ? "r" : "-");
printf( (lstruct.st_mode & S_IWOTH) ? "w" : "-");
printf( (lstruct.st_mode & S_IXOTH) ? "x\t" : "-\t");

printf("%ld", (long) lstruct.st_nlink);
if(loption!=2)
printf("\t%ld", (long) lstruct.st_uid);
if(ooption==0)
printf("\t%ld", (long) lstruct.st_gid);
printf("\t%lld", (long long) lstruct.st_size);
printf("\t%s", ctime(&lstruct.st_atime));
printf("\t%s", ctime(&lstruct.st_mtime));
printf("\t%s", ctime(&lstruct.st_ctime));
printf("\t%ld", (long) lstruct.st_blksize);
}

```

## 2.3 Fazit

Um unsere Programm zu testen wechseln wir ins *testDir* Ordner hier rufen wir unsere **mys** Programm ohne Parameter auf:

```
~$ ./mys
```

und erhalten Ausgabe:

```

input path: /home/xenia/testDir/
resolved_path: /home/xenia/testDir
ausfuerbar
symlinkToFileOwnerUserGroupTest
file.cpp
file.c
dir1
file2.c
fileOwnerUserGroupUser
file1
fileOwnerUserGroupTest

```

wie man sieht unsere Programm gibt zuerst vollen Pfad zu dem Ordner für den **mys** ausgeführt wird, dies wurde für testzwecke gemacht

Bei diesen Test haben wir kein Pfad angegeben, deswegen wird das `myls` für aktuellen Verzeichniss ausgeführt.

Die nächsten Zeilen werden alle Dateien im Ordner `testDir` Angezeigt (ausgenommen versteckte Dateien)

als nächstes rufen wir `myls` mit Parameter `-a` auf:

```
~$ ./mysls -a
```

so werden auch versteckte Dateien aufgelistet:

```
..
ausfuerbar
symlinkToFileOwnerUserGroupTest
file.cpp
file.c
dir1
file2.c
fileOwnerUserGroupUser
file1
.
fileOwnerUserGroupTest
.dirHidden
.hiddenFile
```

Nächste test Parameter `-l`:

```
~$ ./mysls -l
```

in der Ausgabe sehen wir dass `myls` zusätzliche information zu jeden File ausgibt. Außerdem werden Verzeichnisse rot markiert und dateien mit Endung `.c` grün gefärbt.

Ausgabeformat ist in der reihenfolge:



```
rw-rw-r-- 1 1000 1000 0 Wed Jun 8 20:30:38 2016
Wed Jun 8 20:31:02 2016
4096 ausfuerbar
rw-rw-r-- 1 1000 1000 22 Wed Jun 8 20:38:13 2016
Wed Jun 8 20:38:12 2016
4096 symlinkToFileOwnerUserGroupTest
rw-rw-r-- 1 1000 1000 0 Wed Jun 8 19:40:24 2016
Wed Jun 8 19:40:24 2016
4096 file.cpp
rw-rw-r-- 1 1000 1000 0 Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
4096 file.c
rw-rw-r-- 2 1000 1000 4096 Wed Jun 8 19:41:02 2016
Wed Jun 8 19:41:01 2016
4096 dir1
rw-rw-r-- 1 1000 1000 0 Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
4096 file2.c
rw-rw-r-- 1 1000 1000 0 Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
4096 fileOwnerUserGroupUser
rw-rw-r-- 1 1000 1000 0 Wed Jun 8 19:38:43 2016
Wed Jun 8 19:38:43 2016
4096 file1
rw-rw-r-- 1 1000 1001 0 Wed Jun 8 19:38:02 2016
Wed Jun 8 20:24:52 2016
4096 fileOwnerUserGroupTest
xenia@xenia-IdeaPad:~/testDir$ ^C
```

Nächste Test Parameter `-g` (kein owner id):

```
~$ ./mysls -g
```

Wir sehen deutlich beim File fileOwnerUserGroupTest kein owner id(1000) angezeigt wird

```

4096 file1-
rwxrwxrwx 1 1000 22 Wed Jun 8 20:38:13 2016
Wed Jun 8 20:38:12 2016
Wed Jun 8 20:38:12 2016
4096 symlinksToFileOwnerUserGroupTest
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:40:24 2016
Wed Jun 8 19:40:24 2016
Wed Jun 8 19:40:24 2016
4096 file.cpp
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
4096 file.c
rwxrwxr-x 2 1000 4096 Wed Jun 8 19:41:02 2016
Wed Jun 8 19:41:01 2016
Wed Jun 8 19:41:01 2016
4096 dir1
-rw-rw-r-- 1 1000 0 Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
4096 file2.c
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
4096 fileOwnerUserGroupUser
-rw-rw-r-- 1 1000 6 Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
4096 file1
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:38:02 2016
Wed Jun 8 19:38:02 2016
Wed Jun 8 20:24:52 2016
4096 fileOwnerUserGroupTest

```

Nächste Test Parameter -g (kein group id):

~\$ ./myls -o

Wir sehen deutlich beim File fileOwnerUserGroupTest kein group id(1001) angezeigt wird

```

Wed Jun 8 19:40:24 2016
4096 file.cpp
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
4096 file.c
rwxrwxr-x 2 1000 4096 Wed Jun 8 19:41:02 2016
Wed Jun 8 19:41:01 2016
Wed Jun 8 19:41:01 2016
4096 dir1
-rw-rw-r-- 1 1000 0 Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
4096 file2.c
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
4096 fileOwnerUserGroupUser
-rw-rw-r-- 1 1000 6 Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
4096 file1
-rw-rw-r-- 1 1000 0 Wed Jun 8 19:38:02 2016
Wed Jun 8 19:38:02 2016
Wed Jun 8 20:24:52 2016
4096 fileOwnerUserGroupTest

```

als nächstes testen wir alle Parameter -algo:

~\$ ./myls -algo

hier wird weder owner id noch group id angezeigt

```

rw-rw-r-- 1 0 Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
4096 file.c
rwxrwxr-x 2 4096 Wed Jun 8 19:41:02 2016
Wed Jun 8 19:41:01 2016
Wed Jun 8 19:41:01 2016
4096 dir1
rw-rw-r-- 1 0 Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
4096 file2.c
rw-rw-r-- 1 0 Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
4096 fileOwnerUserGroupUser
rw-rw-r-- 1 6 Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
4096 file1
rwxrwxr-x 4 4096 Wed Jun 8 21:53:29 2016
Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
4096
rw-rw-r-- 1 0 Wed Jun 8 19:38:02 2016
Wed Jun 8 19:38:02 2016
Wed Jun 8 20:24:52 2016
4096 fileOwnerUserGroupTest
rwxrwxr-x 2 4096 Wed Jun 8 19:41:53 2016
Wed Jun 8 19:41:52 2016
Wed Jun 8 19:41:52 2016
4096 .dirHidden
rw-rw-r-- 1 0 Wed Jun 8 19:39:54 2016
Wed Jun 8 19:39:54 2016
Wed Jun 8 19:39:54 2016
4096 .hiddenFile

```

und alle Parameter in beliebige Reihenfolge:

~\$ ./mys -oag

```

rw-rw-r-- 1 0 Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
Wed Jun 8 19:40:05 2016
4096 file.c
rwxrwxr-x 2 4096 Wed Jun 8 19:41:02 2016
Wed Jun 8 19:41:01 2016
Wed Jun 8 19:41:01 2016
4096 dir1
rw-rw-r-- 1 0 Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
Wed Jun 8 20:17:22 2016
4096 file2.c
rw-rw-r-- 1 0 Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
Wed Jun 8 19:37:17 2016
4096 fileOwnerUserGroupUser
rw-rw-r-- 1 6 Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
4096 file1
rwxrwxr-x 4 4096 Wed Jun 8 21:53:29 2016
Wed Jun 8 21:53:24 2016
Wed Jun 8 21:53:24 2016
4096
rw-rw-r-- 1 0 Wed Jun 8 19:38:02 2016
Wed Jun 8 19:38:02 2016
Wed Jun 8 20:24:52 2016
4096 fileOwnerUserGroupTest
rwxrwxr-x 2 4096 Wed Jun 8 19:41:53 2016
Wed Jun 8 19:41:52 2016
Wed Jun 8 19:41:52 2016
4096 .dirHidden
rw-rw-r-- 1 0 Wed Jun 8 19:39:54 2016
Wed Jun 8 19:39:54 2016
Wed Jun 8 19:39:54 2016
4096 .hiddenFile

```