```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include <stddef.h>

//Variablen
int defrag(void);

char *pfadcpy1;
char *pfadcpy2;
char structure[11]=".structure";
char store[11]=".store";

//Öffentliche Strukturen
struct d
{
int64_t max; /*Dient zur Initialisierung des Dateinamen (Maximale Länge aller eingegebenen
Dateinamen)*/
int64_t blockz;
int64_t blockc;
int blockbelegt; /*0 = "Block noch frei", 1 = "Block belegt"*/
int64_t blocknummer;
int64_t dateigroesse;
char dateiname[];
} *dateiinfo;

////////////////////////////////////////////////////////////////////////////Funktion 3.1
"create"//////////////////////////////////////////////////////////////////////
int create(char *blocksize, char *blockcount)
{

int64_t bz;
int64_t bc;

bz = strtoull(blocksize,0,10);
bc = strtoull(blockcount,0,10);

// Erstellung der Datei .store im Modus "w+" und Überprüfung ob diese schon vorhanden ist.
FILE *speich;
speich = fopen(pfadcpy2,"r");
if(speich != NULL)
{
printf("Ein VFS mit dem gegebenen Namen existiert bereits.\n");
return 3;
}
else
{
speich = fopen(pfadcpy2,"w+");
        if(speich == NULL)
        {
        printf("Das Anlegen des VFS war nicht moglich - Aus Grunden,die das Hostsystem und nicht Ihr
Programm verursacht hat!\n");
        return 1;
        }
}

//Schreibe bc mal ein Objekt der größe bz Bytes in den Stream "speich".
char *puffer;
puffer = malloc(bz*sizeof(char));
int64_t u = 0;
memset(puffer,' ', bz*sizeof(char));
while(u < bc)
{
fread(puffer, bz*sizeof(char), 1, speich);
fwrite(puffer, bz*sizeof(char), 1, speich);
u = u+1;
}
free(puffer);
puffer = NULL;
fclose(speich);
```

```
//Initialisierung der Struktur mit Blockgröße und Blockanzahl.
dateiinfo = malloc(sizeof(struct d) + sizeof(char));
memset(dateiinfo,' ', sizeof(struct d)+ sizeof(char));
dateiinfo->max = 1;
strcpy(dateiinfo->dateiname,"");
dateiinfo->blockz = bz;
dateiinfo->blockc = bc;
dateiinfo->blockbelegt = 0; //Alle unbelegt.

//Erstellung von Verwaltungsstrukturen

FILE *verwalt;
verwalt = fopen(pfadcpy1,"w+");
if(verwalt == NULL)
{
return 1;
}

int64_t s = 0;
while(s < bc)
{
fwrite(dateiinfo, sizeof(struct d) + sizeof(char), 1, verwalt);
s = s+1;
}

free(dateiinfo);
fclose(verwalt);

return 0;
}

////////////////////////////////////////////////////////////Funktion 3.2
"add"/////////////////////////////////////////////////////////////////////////////
int add(char *source, char *target)
{
FILE *temp;
FILE *verwalt;
FILE *speich;

verwalt = fopen(pfadcpy1, "r+");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}
speich = fopen(pfadcpy2, "r+");
if (speich == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}
//Ermittlung der Länge von target
int64_t nl = strlen(target)+1;

dateiinfo = malloc(sizeof(struct d));//Ermittlung der länge des Dateinamens
fread(dateiinfo,sizeof(struct d),1,verwalt);
int64_t max = dateiinfo->max;
free(dateiinfo);
rewind(verwalt);
dateiinfo = malloc(sizeof(struct d)+ max*sizeof(char)); //Neubeschaffung angemessener Speichergröße
für struct d + länge von Dateiname*sizeof(char)
fread(dateiinfo,sizeof(struct d)+ max*sizeof(char),1,verwalt);
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
rewind(verwalt);

//Öffnung der Quelle
temp = fopen(source,"r");
if(temp == NULL)
{
printf("Die Quelldatei existiert nicht.");
return 13; //Sollte die Datei nicht vorhanden sein
```

```
}

//Ist die Datei vielleicht schon im VFS?
char *o;
o = malloc(sizeof(struct d)+ max*sizeof(char));
int catch = 0;
while(fread(o,sizeof(struct d)+ max*sizeof(char),1,verwalt) == 1)
{
if(strcmp(dateiinfo->dateiname, target) == 0)
{
catch++;
}
}
if(catch != 0)
{
free(o);
printf("Eine Datei mit diesem Dateinamen existiert bereits");
return 11; //Schon vorhanden
}
free(o);
rewind(verwalt);

//Die Dateigröße in Bytes sowie die Blockanzahl müssen ermittelt und gespeichert werden
int64_t groesse = 0;
char *g;
g = malloc(sizeof(char));
while(fread(g,sizeof(char),1,temp) == 1)
{
groesse = groesse+1;
}
free(g);
int64_t n;
n = (groesse+bz-1)/bz;

//printf("Die Eingelesene Datei ist %lld Bytes groß.\n", groesse);
//printf("Für die Eingelesene Datei werden %lld Blöcke benötigt \n", n);

//Und wie viele Blöcke sind belegt???
int64_t zaehler = 0;
int64_t maxspeich = bc;
int64_t i = 0;
int64_t pos = 0;
while(fread(dateiinfo,sizeof(struct d)+ max*sizeof(char),1,verwalt) == 1)
{
if(dateiinfo->blockbelegt == 0)
{
i = i+1;
zaehler = zaehler+1;

                if(zaehler == n)
                {
                maxspeich = zaehler;
                pos = i-maxspeich;
                break;
                }

}
if(dateiinfo->blockbelegt == 1)
{
i = i+1;
maxspeich = zaehler;
pos = i-maxspeich;
zaehler = 0;
}
if(i == bc && (dateiinfo->blockbelegt == 0) )
{
maxspeich = zaehler;
pos = i - maxspeich;
}
}

//Ausgabe 2 Möglichkeiten. 1. Es existiert ein Maximum -> Nicht genügend Speicherplatz, 2. Es
```

```
existiert ein Minimum -> Datei kann geschrieben werden.
if(maxspeich < n)
{
printf("Nicht genugend freier Speicherplatz im VFS.");
return 12; //Rückgabe, wenn weniger zusammenhängende freie Plätze vorhanden als für die Datei
notwendig.
}

//printf("Mindestens %lld zusammenhängende leere Blöcke sind ab Block %lld noch unbelegt\n",
maxspeich, pos);
//printf("Daher kann die Datei aufgenommen werden.\n");

//Hier wird eine komplett neue .structure geschrieben werden, sodass ein größerer Dateiname, sofern
dieser erforderlich ist, hineinpasst.
//
```

```
rewind(verwalt);

if(max < nl)
{

int64_t i = 0;

char *dat;
dat = malloc(bc*(sizeof(struct d)+max*sizeof(char)));
while(i<bc) //Blöcke in das Array aufnehmen
{
fread(dateiinfo,sizeof(struct d)+ max*sizeof(char), 1, verwalt);
memcpy(dat+(i*(sizeof(struct d)+max*sizeof(char))), dateiinfo, sizeof(struct d)+max*sizeof(char));
i = i+1;
}
i = 0; //Rücksetzen für die Rückrichtungsschleife
fclose(verwalt); //Schließen und Neuöffnung zur Neuschreibung
verwalt = fopen(pfadcpy1,"w+");
        if(verwalt == NULL)
        {
        printf("Datei konnte nicht geöffnet werden (w+).");
        return 66;
        }

free(dateiinfo); //Neue Speichergröße wird in der Schleife benötigt
while(i < bc)
{
//Objekt an der Stelle i in den Struct schreiben
dateiinfo = malloc(sizeof(struct d) + nl*sizeof(char));//Objekt größer machen
memset(dateiinfo,' ', sizeof(struct d)+ nl*sizeof(char));
memcpy(dateiinfo, dat+(i*(sizeof(struct d)+max*sizeof(char))) ,sizeof(struct d)+ max*sizeof(char)); //
Kopiere Element der kleinen größe (+max)
dateiinfo->max = nl; //Neue länge der Datei speichern
fwrite(dateiinfo, sizeof(struct d)+ nl*sizeof(char), 1, verwalt);//Schreiben des nun größeren Objekts
in den Stream
free(dateiinfo);
i = i+1;
}
free(dat);
fclose(verwalt);
verwalt = fopen(pfadcpy1,"r+"); //Wiedereröffnung im "r+"-Modus
        if(verwalt == NULL)
        {
        printf("Datei konnte nicht geöffnet werden (r+).");
        return 66;
        }
dateiinfo = malloc(sizeof(struct d)+ nl*sizeof(char));
}
//
```

```
//Wenn genügend Plätze da sind, schreibe die Datei ab Block "pos" in den Store
free(dateiinfo);
rewind(verwalt);
dateiinfo = malloc(sizeof(struct d));
//Bedinungsunabhängigen neuen oder alten max holen
```

```
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t newmax = dateiinfo->max; //richtig
free(dateiinfo);
rewind(verwalt);
rewind(temp);

//Dateiinfoobjekt mit richtiger Größe öffnen
dateiinfo = malloc(sizeof(struct d) + newmax*sizeof(char));
memset(dateiinfo,' ', sizeof(struct d)+ newmax*sizeof(char));
//Dateiinfo wird mit den Daten der Datei beschrieben
fread(dateiinfo,sizeof(struct d) + newmax*sizeof(char), 1, verwalt);
dateiinfo->blockbelegt = 1;

strcpy((dateiinfo->dateiname),target); //Neuen Namen speichern
dateiinfo->dateigroesse = groesse;
rewind(verwalt);

//Zeigersetzung für das Schreiben ab der ersten freien Position
fseek(speich, pos*bz, SEEK_SET);
fseek(verwalt,pos*(sizeof(struct d) + newmax*sizeof(char)), SEEK_SET);

char *p;
p = malloc(bz*sizeof(char));
int64_t m = 0;
int64_t blockpos = pos; //Zähler, um Dateiinfo zwischendurch mit der aktuellen Blocknummer zu
initialisieren

        while( m < n)
        {
        memset(p,' ', bz*sizeof(char));
        fread(p, bz*sizeof(char), 1, temp);
        fwrite(p, bz*sizeof(char), 1, speich );
        dateiinfo->blocknummer = blockpos;
        fwrite(dateiinfo, sizeof(struct d)+ newmax*sizeof(char), 1, verwalt);
        m = m+1;
        blockpos = blockpos+1;
        }
free(p);
free(dateiinfo);
fclose(speich);
fclose(verwalt);
fclose(temp);

}//Funktion Ende


////////////////////////////////////////////////////////////////////////////Funktion 3.3
"get"/////////////////////////////////////////////////////////////////////////////////

int get(char *source, char *output)
{

FILE *temp;
FILE *verwalt;
FILE *speich;

verwalt = fopen(pfadcpy1, "r+");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}
speich = fopen(pfadcpy2, "r+");
if (speich == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}

//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
```

```
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
//printf("gibt zurück max %lld, bc %lld, bz %lld, blockbelegt %d,blocknummer %lld, dateigroesse %lld
\n", dateiinfo->max, bc, bz,dateiinfo->blockbelegt,dateiinfo->blocknummer,dateiinfo->dateigroesse);
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////

////////Kopiervorgang von speich in temp Stream ;)
int64_t bzaehl = 0;
char *p;
p = malloc(sizeof(char));
int64_t k = 0;
int hilf = 0;  //Schalter klappt um, wenn die Datei im VFS gefunden wurde - Sodass nur einmal eine
neue Datei erstellt wird

while(fread(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt) == 1)
{
        if(strcmp(dateiinfo->dateiname, source) == 0)
        {
        fseek(speich,k*bz,SEEK_SET);

        while(bzaehl < dateiinfo->dateigroesse)
                {
                //memset(p,' ', 1);
                fread(p, sizeof(char), 1, speich);

                            if(hilf == 0)
                            {
                            temp = fopen(output, "w+");
                                        if(temp == NULL)
                                        {
                                        printf("Die Zieldatei im Host-Dateisystem konnte nicht
erstellt werden.");
                                        return 30;
                                        }
                                hilf++;
                            }


                fwrite(p, sizeof(char), 1, temp);
                bzaehl = bzaehl + 1;
                }
        fclose(temp);
        free(dateiinfo);
        free(p);
        return 0;
        }
k++;
}

free(dateiinfo);
free(p);

//Wurde eine Datei gefunden?
if(hilf == 0)
{
printf("Die Datei existiert nicht im VFS.");
return 21; //Datei existiert nicht im VFS
}

fclose(temp);
fclose(verwalt);
fclose(speich);
}


////////////////////////////////////////////////////////////////////////////////Funktion 3.4
"del"//////////////////////////////////////////////////////////
int del(char *target)
```

```
{

FILE *verwalt;
FILE *speich;
verwalt = fopen(pfadcpy1, "r+");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}
speich = fopen(pfadcpy2, "r+");
if(speich == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}

//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
//printf("gibt zurück max %lld, bc %lld, bz %lld, blockbelegt %d,blocknummer %lld, dateigroesse %lld
\n", dateiinfo->max, bc, bz,dateiinfo->blockbelegt,dateiinfo->blocknummer,dateiinfo->dateigroesse);
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
memset(dateiinfo,' ', sizeof(struct d)+ max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////


char *p;//Puffer für Speicherdatei
p = malloc(bz*sizeof(char));
memset(p,' ',bz*sizeof(char));
int64_t k = 0;
int schalter = 0;
while(fread(dateiinfo,sizeof(struct d)+max*sizeof(char), 1, verwalt) == 1)
{
        if(strcmp(dateiinfo->dateiname, target) == 0)
        {
        dateiinfo->blockbelegt = 0;
        dateiinfo->dateigroesse = 0;
        dateiinfo->blocknummer = 0;
        strcpy(dateiinfo->dateiname, "");
        fseek(verwalt,k*(sizeof(struct d)+max*sizeof(char)),SEEK_SET);
        fwrite(dateiinfo, sizeof(struct d)+max*sizeof(char), 1, verwalt);
        fseek(speich, k*bz*sizeof(char),SEEK_SET);
        fwrite(p, bz*sizeof(char), 1, speich);
        schalter = 1;
        }
k++;
}
free(p);

if(schalter == 0)
{
free(dateiinfo);
printf("Die Datei existiert nicht im VFS.");
return 21;
}

//_______Reduktion der
Strukturdateigröße_____________________________________________________________________________
rewind(verwalt);
//Ermittlung alter größter länge
int64_t u = 0;
int64_t oldmax = -1;
while(fread(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt) == 1)
{
u = strlen(dateiinfo->dateiname);
```

```
        if(u >= oldmax)
        {
        oldmax = u + 1; // In oldmax bleibt höchste länge hängen
        }
}

if(oldmax == max) //Falls weitere Elemente mit der alten größe max vorhanden sind
{
fclose(speich);
fclose(verwalt);
free(dateiinfo);
defrag();
return 0;
}

rewind(verwalt);
//Neuschreiben der Strukturdatei
int64_t i = 0;
char *dat;
dat = malloc(bc*(sizeof(struct d) + max*sizeof(char)));
while(i<bc) //Blöcke in das Array aufnehmen
{
fread(dateiinfo,sizeof(struct d) + max*sizeof(char), 1, verwalt);
memcpy(dat+(i*(sizeof(struct d) + max*sizeof(char))), dateiinfo, sizeof(struct d)+max*sizeof(char));
i = i+1;
}
i = 0; //Rücksetzen für die Rückrichtungsschleife

fclose(verwalt); //Schließen und Neuöffnung zur Neuschreibung
verwalt = fopen(pfadcpy1,"w+");
        if(verwalt == NULL)
        {
        return 66;
        }

free(dateiinfo); //Neue Speichergröße wird in der Schleife benötigt
rewind(verwalt);
while(i < bc)
{
//Objekt an der Stelle i in den Struct schreiben
dateiinfo = malloc(sizeof(struct d) + oldmax*sizeof(char));//Objekt kleiner machen
memset(dateiinfo,' ', sizeof(struct d)+ oldmax*sizeof(char));
memcpy(dateiinfo, dat+(i*(sizeof(struct d) + max*sizeof(char))) ,sizeof(struct d)+oldmax*sizeof
(char)); //Kopiere Element der kleinen größe (oldmax) aus größerem Objekt in dateiinfo
dateiinfo->max = oldmax; //Neue Länge der Datei speichern
fwrite(dateiinfo, sizeof(struct d) + oldmax*sizeof(char), 1, verwalt);//Schreiben des neuen Objekts in
den Stream
free(dateiinfo);
i = i+1;
}
//
```

```
free(dat);
fclose(verwalt);
fclose(speich);
defrag();

return 0;
}//Funktion Ende

////////////////////////////////////////////////////////////////////////////////Funktion 3.5
"free"//////////////////////////////////////////////////////////////////////////////
int freee()
{
FILE *verwalt;
verwalt = fopen(pfadcpy1, "r+");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}
```

```
//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////


//Zähle k jedes mal hoch, wenn ein Block unbelegt ist
int64_t k = 0;
while(fread(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt) == 1)
{
        if(dateiinfo->blockbelegt == 0)
        {
        k = k+1;
        }
}
fclose(verwalt);
free(dateiinfo);
int64_t bytes = 0;
bytes = bz*k;
printf("%lld", bytes);

return 0;
}//Funktion Ende




//////////////////////////////////////////////////////////////////Funktion 3.6
"used"/////////////////////////////////////////////////////////////////////////
int used()
{
FILE *verwalt;
verwalt = fopen(pfadcpy1, "r");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}

//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
//printf("gibt zurück max %lld, bc %lld, bz %lld, blockbelegt %d,blocknummer %lld, dateigroesse %lld
\n", dateiinfo->max, bc, bz,dateiinfo->blockbelegt,dateiinfo->blocknummer,dateiinfo->dateigroesse);
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////

//Zähle k jedes mal hoch, wenn ein Block unbelegt ist
int64_t k = 0;
while(fread(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt) == 1)
{
        if(dateiinfo->blockbelegt == 1)
        {
        k = k+1;
        }
}
fclose(verwalt);
free(dateiinfo);
int64_t bytes = 0;
bytes = bz*k;
```

```
printf("%lld", bytes);

return 0;
}//Funktion Ende


////////////////////////////////////////////////Funktion 3.7
"list"//////////////////////////////////////////////////////////////////////
int list()
{
char *prechar;
int b = 0; //boolischer Hilfswert

FILE *verwalt;
verwalt = fopen(pfadcpy1,"r");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}

//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////

prechar = malloc(max*sizeof(char));
int schalter = 0;//Wenn prechar bereits einmal allokiert wurde. Dringend notwendig, da sonst Segfault.
int64_t k = 0; //für dritte if-Abfrage
int64_t h = 0;
while(fread(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt) == 1)
{
        if(k != 0 )
        {

                if(dateiinfo->blockbelegt == 1 && schalter == 1)
                {

                                if(strcmp(dateiinfo->dateiname, prechar) != 0)
                                {
                                b = 0;


                                printf("\n");
                                }
                }
        }

        k = 1;

        if(b == 0 && (dateiinfo->dateigroesse != 0))
        {
        prechar = realloc(prechar, max*sizeof(char));
        strcpy(prechar, dateiinfo->dateiname);
        printf("%s",dateiinfo->dateiname);
        printf(",%lld",dateiinfo->dateigroesse);
        printf(",%lld",(((dateiinfo->dateigroesse)+(dateiinfo->blockz)-1)/dateiinfo->blockz));
        b = 1;
        schalter = 1;
        }

        if(dateiinfo->blockbelegt == 1)
        {
        printf(",%lld", dateiinfo->blocknummer);
        }
```

```
h++;
}
printf("\n");

free(dateiinfo);
free(prechar);
return 0;
}//Funktion Ende

////////////////////////////////////////////////////////////////////Hilfsfunktion tausche() für
defrag//////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////Tausche die a unbelegten Blöcke durch die b
belegten Blöcken von Block pos aus aus.////////////////////////////////////////
int tausche(int64_t a, int64_t b, int64_t pos)
{

FILE *verwalt;
FILE *speich;

verwalt = fopen(pfadcpy1, "r+");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}
speich = fopen(pfadcpy2, "r+");
if (speich == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}

//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
memset(dateiinfo,' ', sizeof(struct d)+ max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////

char *p; //Puffer für die Speicherdatei
p = malloc(bz*sizeof(char));

fseek(verwalt,(pos+a)*(sizeof(struct d) + max*sizeof(char)),SEEK_SET);
fseek(speich, (pos+a)*bz*sizeof(char),SEEK_SET);
int64_t s = 0;
int64_t t = 0;

while(s < b) //Ein Schleifendurchlauf pro belegtem Block, der verschoben werden soll
{
fread(dateiinfo, (sizeof(struct d) + max*sizeof(char)), 1, verwalt);
fseek(verwalt, (pos+s)*(sizeof(struct d) + max*sizeof(char)),SEEK_SET);
dateiinfo->blocknummer = pos+s;
fwrite(dateiinfo, (sizeof(struct d) + max*sizeof(char)), 1, verwalt);
s = s+1;
fseek(verwalt, (pos+a+s)*(sizeof(struct d) + max*sizeof(char)),SEEK_SET);

fread(p, bz*sizeof(char), 1, speich);
fseek(speich, ((pos+t)*bz)*sizeof(char),SEEK_SET);
fwrite(p, bz*sizeof(char), 1, speich);
t = t+1;
fseek(speich, ((pos+a+t)*bz)*sizeof(char),SEEK_SET);
}

//Jetzt müssen die virtuellen leeren Blöcke noch überschrieben werden
strcpy(dateiinfo->dateiname, "");
dateiinfo->blockbelegt = 0;
dateiinfo->blocknummer = 0;
```

```
dateiinfo->dateigroesse = 0;

fseek(verwalt,(pos+b)*(sizeof(struct d) + max*sizeof(char)),SEEK_SET);
fseek(speich, (pos+b)*bz*sizeof(char),SEEK_SET);
int64_t l = 0;
memset(p,' ', bz*sizeof(char));
while(l < a)//So viele Schleifendurchläufe, wie es leere Blöcke gab.
{
fwrite(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt);
fwrite(p, bz*sizeof(char), 1, speich);

l = l+1;
}

free(dateiinfo);
free(p);

fclose(verwalt);
fclose(speich);

}//Funktion Ende



////////////////////////////////////////////////////////////////////////////////Funktion 3.8
"defrag"////////////////////////////////////////////////////////////////////////////////
int defrag()
{

FILE *verwalt;

verwalt = fopen(pfadcpy1, "r+");
if(verwalt == NULL)
{
printf("Das VFS konnte nicht geoffnet werden.");
return 2;
}

//Allgemeine Routine//////////////////////////
dateiinfo = malloc(sizeof(struct d));
fread(dateiinfo,sizeof(struct d), 1, verwalt);
int64_t max = dateiinfo->max;
int64_t bc = dateiinfo->blockc;
int64_t bz = dateiinfo->blockz;
free(dateiinfo);
dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
rewind(verwalt);
////////////////////////////////////////////


char *prechar;
prechar = malloc(max*sizeof(char));
rewind(verwalt);

int schalter = 0;
int64_t a = 0; //Anzahl freier Speicherblöcke vor einer Datei
int64_t b = 0; //Anzahl belegter Speicherblöcke durch die Datei
int64_t n = 0;
int64_t pos = 0;


while(fread(dateiinfo, sizeof(struct d) + max*sizeof(char), 1, verwalt) == 1)
{
        if(schalter == 1)
        {
                if(n == 1)
                {
                        if(strcmp(dateiinfo->dateiname, prechar) != 0)
                        {

                        n = 0;
                        fclose(verwalt);
```

```
                        free(dateiinfo);
                        tausche(a,b,pos);
                        dateiinfo = malloc(sizeof(struct d) + max*sizeof(char));
                        verwalt = fopen(pfadcpy1,"r+");
                        pos = pos+b;
                        a = 0;
                        b = 0;

                        fseek(verwalt, pos*(sizeof(struct d) + max*sizeof(char)), SEEK_SET);

                        continue;
                        }
                }
        }


        if(n == 0 && (dateiinfo->blockbelegt == 1))
        {
        prechar = realloc(prechar, max*sizeof(char));
        strcpy(prechar, dateiinfo->dateiname);
        n = 1;
        schalter = 1; //Dauerschalter
        }

        if(strcmp(dateiinfo->dateiname,"") != 0)
        {
        b = b+1;
        }
        if(strcmp(dateiinfo->dateiname,"") == 0)
        {
        a = a+1;
        }

}//Schleife zu

free(prechar);
free(dateiinfo);
        if(a!=bc)
        {
        tausche(a,b,pos);
        }
}//Funktion zu


////argv[0] == "vfs", argv[1] == "archiv",argv[2] == "operation",argv[3,4] == "parameter",///////////
main-Funktion ////////////////////////////////////////////////////////////////////
int main(int argc, char *argv[])
{
if(argc < 3)
{
return 66;
}

//Initialisierung der Dateipfade von .structure und .store
int i = strlen(argv[1]);
i = i+11;
pfadcpy1 = malloc(i*sizeof(char));
strcpy(pfadcpy1, argv[1]);
pfadcpy2 = malloc(i*sizeof(char));
strcpy(pfadcpy2, argv[1]);
//Anhängen der ".store", ".structure"-Dateiendungen und somit redefinierung der Pfade als Dateinamen
strcat(pfadcpy2,store);
strcat(pfadcpy1,structure);
int h; // Hilfsvariable um Übergeben der Rückgaben an die Ausgabefunktion ret()
int zeichen;

//create-Abfrage
                        if(strcmp(argv[2], "create") == 0)
                        {
                                if(argc < 5)
                                {
                                        return 66;
```

```
                }
        h = create(argv[3], argv[4]);
        free(pfadcpy1);
        free(pfadcpy2);
        return h;
        }


//add-Abfrage
        if(strcmp(argv[2], "add") == 0)
        {
                if(argc < 5)
                {
                        return 66;
                }
        h = add(argv[3], argv[4]);
        free(pfadcpy1);
        free(pfadcpy2);
        return h;
        }

//del-Abfrage
        if(strcmp(argv[2], "del") == 0)
        {
                if(argc < 4)
                {
                        return 66;
                }
        h = del(argv[3]);
        free(pfadcpy1);
        free(pfadcpy2);
        return h;
        }



//defrag-Abfrage
        if(strcmp(argv[2], "defrag") == 0)
        {

                h = defrag();
                free(pfadcpy1);
                free(pfadcpy2);
                return h;
        }

//free-Abfrage
        if(strcmp(argv[2], "free") == 0)
        {

                h = freee();
                free(pfadcpy1);
                free(pfadcpy2);
                return h;
        }


//get-Abfrage
        if(strcmp(argv[2], "get") == 0)
        {
                if(argc < 5)
                {
                        return 66;
                }
        h = get(argv[3], argv[4]);
        free(pfadcpy1);
        free(pfadcpy2);
        return h;
        }

//list-Abfrage
        if(strcmp(argv[2], "list") == 0)
```

```
                        {

                                h = list();
                                free(pfadcpy1);
                                free(pfadcpy2);
                                return h;
                        }

//used-Abfrage
                        if(strcmp(argv[2], "used") == 0)
                        {

                                h = used();
                                free(pfadcpy1);
                                free(pfadcpy2);
                                return h;

                        }


//Abfrage für jede andere Eingabe
                        else
                        {
                                free(pfadcpy1);
                                free(pfadcpy2);
                                return 66;

                        }

pfadcpy1 = NULL;
pfadcpy2 = NULL;
return EXIT_SUCCESS;
}
```