1 Lösungsidee

Es soll eine rekursive Funktion implementiert werden, die die Verzeichnisse rekursiv durchwandert. Diese Funktion soll die unterstützen File Handler Funktionen aufrufen und zwar für jedes besuchtes File und Verzeichnis.
Da es anscheinend nicht möglich ist die Vargs an den nächsten rekursiven Aufruf weiterzuleiten (function foo(int n, …)), sollen diese in einer übergeordneten Funktion, welche die rekursive aufruft, in ein char** Array kopiert werden und dieses Array soll über die rekursiven Aufrufe hinweg durchgeschleust werden.
Fehlerhafte Pfadangaben und/oder Funktionen sollen gehandhabt und den Benutzer über eine Meldung auf der Konsole informiert werden.

Print:
Diese Funktion soll die Dateiattribute (Dateityp, Rechte, Datei- oder Verzeichnisname, Größe, …) auf der Konsole ausgeben. Da nicht verlangt soll hierbei keine Rücksicht auf die Ordnung genommen werden. Die Auswertung der Datei soll über die Standardlib stat.h erfolgen.

Grep:
Diese Funktion soll die Dateien auf einen gegebenen String hin durchsuchen. Da sich in den ersten Versuchen gezeigt hatte, dass das zeilenweise Lesen einer Binärdatei zu Problemen führt soll der Dateityp ermittelt werden und die Suche soll nur in ASCII Text Dateien erfolgen um diesen Problemen zu entgehen. Dies soll über einen System Call erfolgen, wobei mittels `file <filename> | grep ASCII` ermittelt werden soll ob die zur Zeit besuchte Datei eine ASCII Date ist und somit auch zeilenweise gelesen werden kann.
Des Weiteren wurde festgestellt, dass es Textdateien gibt, welche keine Zeilenumbrüche enthalten, Probleme mit der Funktion getLine(…) verursachen, daher sollen die Zeilen mittels fget(…) gelesen werden wobei die maximalle Anzahl der zu lesenden Buchstaben auf 10000 begrenzt werden soll. Diese Funktion soll die gefundene Datei mit der Zeile in der der String gefunden wurde ausgeben. Als Zeilenindex soll ein Integer fungieren, der bei jedem Lesen von maximal 10 000 Buchstaben inkrementiert werden soll. Dies ist ein Workaround aber vorerst nicht anders zu lösen.

1.1 Implementierung

Folgend ist die Implementierung diese Aufgabe angeführt

1.1.1 file_visitor.h

Folgend ist der Source der Header Datei angeführt, welche diese Applikation spezifiziert.

```c
/*
 * file_handler.h
 *
 *  This header file specifies the functions for the handling of files.
 *
 *  Created on: Nov 1, 2014
 *      Author: cchet
 */

#ifndef FILE_HANDLER_H_
#define FILE_HANDLER_H_

#include <stdarg.h>

/* For comparison of the intended function */
#define PRINT_FILE_PROPS "print"
#define SEARCH_FILE "grep"

/**
 * This is the specification for the function which handles the
 * files visited by the calling function.
 *
 * @param
 *      const char* pathname: the fully qualified path to the visited file.
 * @param
 *      const struct stat stat: the current file stats
 * @param
 *      const n the count of the arguments in the array
 * @param
 *      array the arguments array
 */
typedef void (*Visitor)(const char *pathname, const struct stat stat, const int n, char**
array);

/**
 * Visits all files and directories which are children of the given path.
 *
 * @param
 *      const char* dirname: the directory to visit its content
 * @param
 *      const Visitor: the function pointer to the visitor function which handles the visited
file
 * @param
 *      const n: the count of vargs arguments
 * @param
 *      The varargs for the visitor function
 */
void traverse(const char *dirname, const Visitor visitor, const int n, ...);

/**
 * Prints all the of the file attributes.
 * Follows Visitor type function signature specification.
 *
 * @param
 *      const char* pathname: the fully qualified path to the visited file.
 * @param
 *      const struct stat stat: the current file stats
 * @param
 *      const n the count of the arguments in the array
 * @param
 *      array the arguments array
 */
void print(const char *pathname, const struct stat stat, const int n, char** array);

/**
 * Searches a text in the found ascii files.
 *
 * @param
 *      const char* filename: the current visited file name
 * @param
 *      const struct stat stat: the current file stats
 * @param
 *      const n the count of the arguments in the array
 * @param
 *      array the arguments array
 */
void grep(const char *filename, const struct stat stat, const int n, char** array);

#endif /* FILE_HANDLER_H_ */
```

1.1.2 file_visitor.c

Folgend ist die Implementierung der Header Datei file_visitor.h angeführt.

```
/*
 * file_visitor.c
 *
 *  This is the implementation of the file_visitor. h specification.
 *
 *  Created on: Nov 1, 2014
 *      Author: cchet
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <dirent.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <syscall.h>
#include "file_visitor.h"
#include "common.h"

/* Avoid memory corruption when files have too long line length */
#define MAX_LINE_LENGTH 10000

/**
 * Builds the fully qualified path to the visited file.
 *
 * @param
 *      char* curPath: the current visited path
 * @param
 *      char* file: the current file within the current visited path
 * @return
 *      char*: the build fully qualified path to the file
 */
static char* buildFullyQualifiedPath(const char* curPath, const char* file) {
    char* path = NULL;
    bool separatorPresent = false;
    int curPathLen;

    curPathLen = strlen(curPath);

    /* Check for '/' prefix on current path */
    if (curPath[curPathLen - 1] == '/') {
        separatorPresent = true;
    } /* if */

    path = (char*) malloc(
            (curPathLen + strlen(file))
                    + (separatorPresent ? 0 : 1) * sizeof(char));

    if (path == NULL) {
        printf("Could not allocate memory for path variable !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    /* Avoid characters in fresh allocated array, which happened sometimes */
    path[0] = '\0';
    strcat(path, curPath);

    /* Check for '/' prefix on file name */
    if ((!separatorPresent) && (file[0] != '/')) {
        strcat(path, "/");
    } /* if */

    strcat(path, file);
    return path;
} /* buildFullyQualifiedPath */

/**
 * Checks if the current file is a ASCII text file
 *
 * @param
 *      char* filename: the filename to check for ASCII file
 * @retrun
 *      bool: true if the file is a ASCII file false otherwise
 */
static bool isAsciiTextFile(const char* filename) {
    int sysres;
    char* prefixCommand = "file '\0";
    char* suffixCommand = "' | grep ASCII > /dev/null\0";
    char* fullFileCommand;

    fullFileCommand = (char*) malloc(
            strlen(filename) + strlen(prefixCommand) + strlen(suffixCommand)
                    + 1);
    /* seems that there are already characters present event after new allocation */
    fullFileCommand[0] = '\0';

    if (fullFileCommand == NULL) {
        printf(
                "Could not allocate memory for system command character array !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    /* Build full command string */
    strcat(fullFileCommand, prefixCommand);
    strcat(fullFileCommand, filename);
    strcat(fullFileCommand, suffixCommand);

    /* perform system call */
    sysres = system(fullFileCommand);

    /* free memory */
    free(fullFileCommand);
    fullFileCommand = NULL;

    return sysres == 0 ? true : false;
} /* isAsciiTextFile */

static void walkDir(const char *file, const Visitor visitor, const int n,
        char** vargs) {
    DIR* directory = NULL;
    struct stat fAttr;
    struct dirent* dirAttr = NULL;

    /* check for NULL dirName */
    if (file == NULL) {
        printf("Cannot visit NULL directory !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    /* get file attributes */
    if (stat(file, &fAttr) == -1) {
        printf("Error(%d) reading file attributes from file: '%s' !!!\n", errno,
                file);
        return;
    } /* if */

    /* visit file */
    visitor(file, fAttr, n, vargs);

    /* if directory */
    if ((S_ISDIR(fAttr.st_mode))) {
        if ((directory = opendir(file)) == NULL) {
            printf("Error reading directory: '%s'\n", file);
            return;
        } /* if */
        while ((dirAttr = readdir(directory)) != NULL) {
            /* Exclude the UNIX directory entries '.' '..' which would lead to stack overflow
*/
            if ((strcmp(dirAttr->d_name, "..") > 0)) {
                walkDir(buildFullyQualifiedPath(file, dirAttr->d_name), visitor,
                        n, vargs);
            } /* if */
        } /* while */
        closedir(directory);
    } /* if */
} /* walkDir */

void traverse(const char *dirname, const Visitor visitor, const int n, ...) {
    va_list vargs;
    char** args = NULL;
    char* temp = NULL;
    int i;
    int j, argSize;
    va_start(vargs, n);

    if (n > 0) {
        i = 0;
        args = (char**) malloc(n * sizeof(char**));
        if (args == NULL) {
            printf("Could not allocate memory for arg array !!!\n");
            exit(EXIT_FAILURE);
        } /* if */
        /* copy vargs to array so that t can be passed through the recursive calls. */
        while (i < n) {
            temp = va_arg(vargs, char*);
            argSize = sizeof(temp);
            args[i] = malloc(sizeof(temp));
            for (j = 0; j < argSize; ++j) {
                args[i][j] = temp[j];
            } /* for */
            i++;
        } /* while */
    } /* if */
    walkDir(dirname, visitor, n, args);

    if (n > 0) {
        for (i = 0; i < n; ++i) {
            free(args[i]);
        }
        free(args);
    }
} /* traverse */

/* ############## Visitor implementations ############## */
void print(const char *pathname, const struct stat stat, const int n,
        char** array) {
    char* permissions;

    permissions = (char*) malloc(11 * sizeof(char));
    if (permissions == NULL) {
        printf("Could not allocate memory for permission string !!!\n");
        exit(EXIT_FAILURE);
    } /* if */

    /* build permission string with bitmask to file properties */
    permissions[0] = (S_ISDIR(stat.st_mode)) ? 'd' :
                     (S_ISLNK(stat.st_mode)) ? 'l' : '-';
    permissions[1] = (stat.st_mode & S_IRUSR) ? 'r' : '-';
    permissions[2] = (stat.st_mode & S_IWUSR) ? 'w' : '-';
    permissions[3] = (stat.st_mode & S_IXUSR) ? 'x' : '-';
    permissions[4] = (stat.st_mode & S_IRGRP) ? 'r' : '-';
    permissions[5] = (stat.st_mode & S_IWGRP) ? 'w' : '-';
    permissions[6] = (stat.st_mode & S_IXGRP) ? 'x' : '-';
    permissions[7] = (stat.st_mode & S_IROTH) ? 'r' : '-';
    permissions[8] = (stat.st_mode & S_IWOTH) ? 'w' : '-';
    permissions[9] = (stat.st_mode & S_IXOTH) ? 'x' : '-';
    permissions[10] = '\0';
    /* print the unix like file entry */
    printf("%s %d %d %s%s\n", permissions, stat.st_uid, stat.st_gid,
            ctime(&stat.st_mtime), pathname);

    free(permissions);
} /* print */

void grep(const char *filename, const struct stat stat, const int n,
        char** array) {
    FILE* file = NULL;
    int lines = 1;
    char line[MAX_LINE_LENGTH];
    char* pattern;

    /* search only in regular files */
    if (S_ISREG(stat.st_mode) == 0) {
        return;
    } /* if */

    /* Only search on ASCII files */
    if (isAsciiTextFile(filename)) {

        /* get argument */
        if (n > 0) {
            pattern = array[0];
        } else {
            pattern = "";
        } /* if */

        file = fopen(filename, "r");

        if (file == NULL) {
            printf("Could not open file: '%s' !!!\n", filename);
            return; /* no need to break program */
        } /* if */

        while (fgets(line, MAX_LINE_LENGTH, file) != NULL) {
            if (strstr(line, pattern) != NULL) {
                printf("%s:%d: %s", filename, lines, line);
            } /* if */
            lines++;
        } /* while */

        fclose(file);
    } /* if */
} /* grep */
```

1.1.3 main.c

Folgend ist die Implementierung des Main Programms angeführt.

```c
/*
 * main.c
 *
 *  This source file represents the main entry to this application.
 *
 *  Created on: Nov 1, 2014
 *      Author: cchet
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <dirent.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <syscall.h>
#include "file_visitor.h"
#include "common.h"
#include "file_visitor.h"

/**
 * Answers the question if the given directory name is a valid directory.
 *
 * @param
 *      const char* filename: the filename to validate
 */
static bool isValidFile(const char* filename) {
    DIR* folder = NULL;
    FILE* file = NULL;
    bool result = false;
    folder = opendir(filename);
    if (folder != NULL) {
        result = true;
        closedir(folder);
    } else {
        file = fopen(filename, "r");
        if (file != NULL) {
            result = true;
            fclose(file);
        } /* if */
    } /* if */
    return result;
} /* isValidDir */

/**
 * The main entry point to this program
 *
 * @param
 *      int argc: the argument count
 * @param
 *      char** argv: the arguments for this program
 */
int main(int argc, char** argv) {
    char* functionName;
    char* directory;

    /* check for function argument */
    if (argc < 3) {
        printf(
                "At least the root directory and function must be given !!! E.g.: ./app.bin ./
-grep NULL");
        exit(EXIT_FAILURE);
    } /* if */

    /* Validate directory */
    if (!isValidFile((directory = argv[1]))) {
        printf("Given directory does not exist !!! dir='%s'", argv[1]);
        exit(EXIT_FAILURE);
    } /* if */
    /* Validate given function name */
    functionName = argv[2] + 1;
    if (strcmp(functionName, PRINT_FILE_PROPS) == 0) {
        traverse(directory, &print, 0, NULL);
    } else if (strcmp(functionName, SEARCH_FILE) == 0) {
        if (argc < 4) {
            printf("Missing search pattern !!!");
            exit(EXIT_FAILURE);
        } /* if */
        traverse(directory, &grep, 1, argv[3]);
    } else {
        printf("Function not supported !!! function=%s", functionName);
        exit(EXIT_FAILURE);
    } /* if */
    return EXIT_SUCCESS;
} /* main */
```

1.1.4 Makefile

Folgend ist die Makefile angeführt, mit welcher diese Applikation gebaut werden kann.

```
CC = gcc
CFLAGS = -ansi -g -Og -ansi -pedantic -Wall -Wextra

LD = gcc
LDFLAGS = -lm

PROGRAM = fv

CFILES = $(wildcard *.c)
OFILES = $(CFILES:.c=.o)

%.o: %.c
	$(CC) $(CFLAGS) -c $<

$(PROGRAM): $(OFILES)
	$(LD) -o $@ $(OFILES) $(LDFLAGS)

clean:
	rm -f $(OFILES) $(PROGRAM)

depend:
	@cat < /dev/null > makedep
	@for i in $(CFILES); do \
	    ($(CC) -MM $$i >> makedep); done
	@echo "/^# BEGIN DEPENDENCIES\$$/+1,\$$d" > edcmds
	@echo "r makedep" >> edcmds
	@echo "wq" >> edcmds
	@cp Makefile Makefile.bak
	@ed -v - Makefile < edcmds
	@rm edcmds makedep
	@echo "# LINE REQUIRED FOR MAKEDEPEND" >> Makefile

# BEGIN DEPENDENCIES
file_visitor.o: file_visitor.c file_visitor.h common.h
main.o: main.c file_visitor.h common.h
# LINE REQUIRED FOR MAKEDEPEND
```

1.1.5 file_visitor_tests.sh

Folgend ist der Source des Testskriptes angeführt, welches die Applikation testet.

```
#!/bin/sh
TEST_DIR=file_visitor_test_dir
RES_DIR=resources
SEARCH_DIR=resources
APP=fv
FUN_GREP=grep
FUN_PRINT=print

###########################
## Create test resources ##
###########################
## Create test folder
echo "\n\n"
if [ -d $TEST_DIR ]
then
    echo "clean test folder '$TEST_DIR'"
    rm -rf $TEST_DIR
else
    echo "create test folder '$TEST_DIR'"
    mkdir $TEST_DIR
fi

## Copy test resources
echo "copying test resources"
cp -R $RES_DIR *.c *.h Makefile $TEST_DIR

## Switch to test dir
cd $TEST_DIR
echo "in directory '$TEST_DIR'"


###########################
## Build application     ##
###########################
echo "\ncompiling sources ..."
echo "> make clean depend fv"
make clean depend fv

###########################
## Tests error handling  ##
###########################
echo "\n\n-------------------------------------"
echo "Test error handling "
echo "-------------------------------------"

echo "\n\nCase 1"
echo "No arguments given"
echo "> ./$APP"
./$APP

echo "\n\nCase 2"
echo "Only directory given"
echo "> ./$APP $SEARCH_DIR"
./$APP $SEARCH_DIR

echo "\n\nCase 3"
echo "grep without pattern"
echo "> ./$APP $SEARCH_DIR -grep"
./$APP $SEARCH_DIR -grep

echo "\n\nCase 4"
echo "invalid dir"
echo "> ./$APP /invalidDir -grep"
./$APP /invalidDir -grep

echo "\n\nCase 5"
echo "invalid file"
echo "> ./$APP /invalidDir/test.txt -grep"
./$APP /invalidDir/test.txt -grep

###########################
## Tests expected grep   ##
```

```
###########################
echo "\n\n-------------------------------------"
echo "Test expected grep  "
echo "-------------------------------------"
echo "\nCase 1"
echo "Finds the test in the test resources"
echo "> ./$APP $SEARCH_DIR -grep char*"
./$APP $SEARCH_DIR -grep char*

echo "\n\nCase 2"
echo "Finds the test in the test resources"
echo "./$APP $SEARCH_DIR -grep @param"
./$APP $SEARCH_DIR -grep @param

echo "\n\nCase 3"
echo "Does not find anything in the test resources"
echo "./$APP $SEARCH_DIR -grep MICH_FINDET_MAN_NICHT"
./$APP $SEARCH_DIR -grep MICH_FINDET_MAN_NICHT

###########################
## Tests expected print  ##
###########################
echo "\n\n-------------------------------------"
echo "Test expected print  "
echo "-------------------------------------"
echo "\nCase 1"
echo "invalid $RE -grep"
./$APP $SEARCH_DIR -print

###########################
## Cleanup tests         ##
###########################
## Delete test resources
echo "\n\nremoving folder '$TEST_DIR'"
cd ..
rm -rf $TEST_DIR
echo "\ntests finished and resources released"
```

1.2 Tests

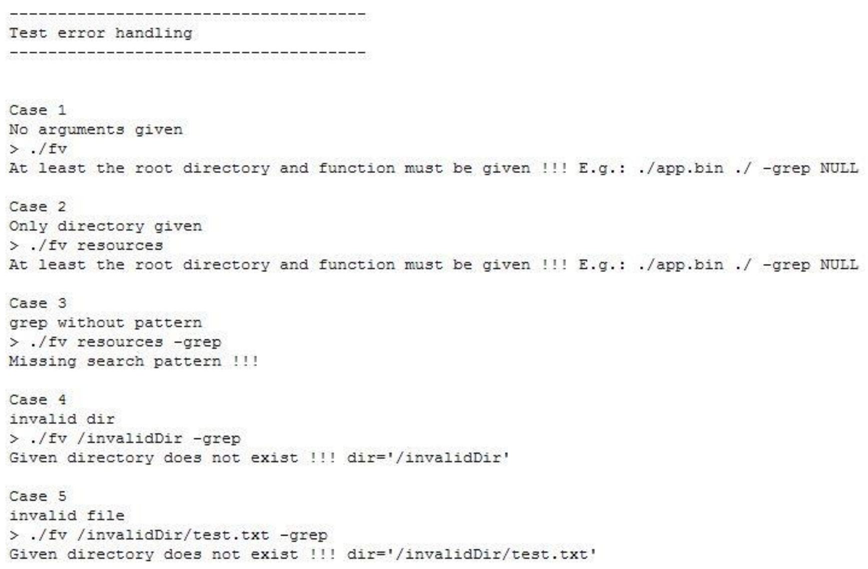
Folgend sind die Test angeführt, welche diese Applikation testen.

Diese Tests wurden in ein Shell Skript implementiert, welches wie folgt aufzurufen ist:

`./file_visitor_tests.sh` (Auf Dateirechte und Ausführbarkeit achten [xr..])

1.2.1 Ungültige Argumente

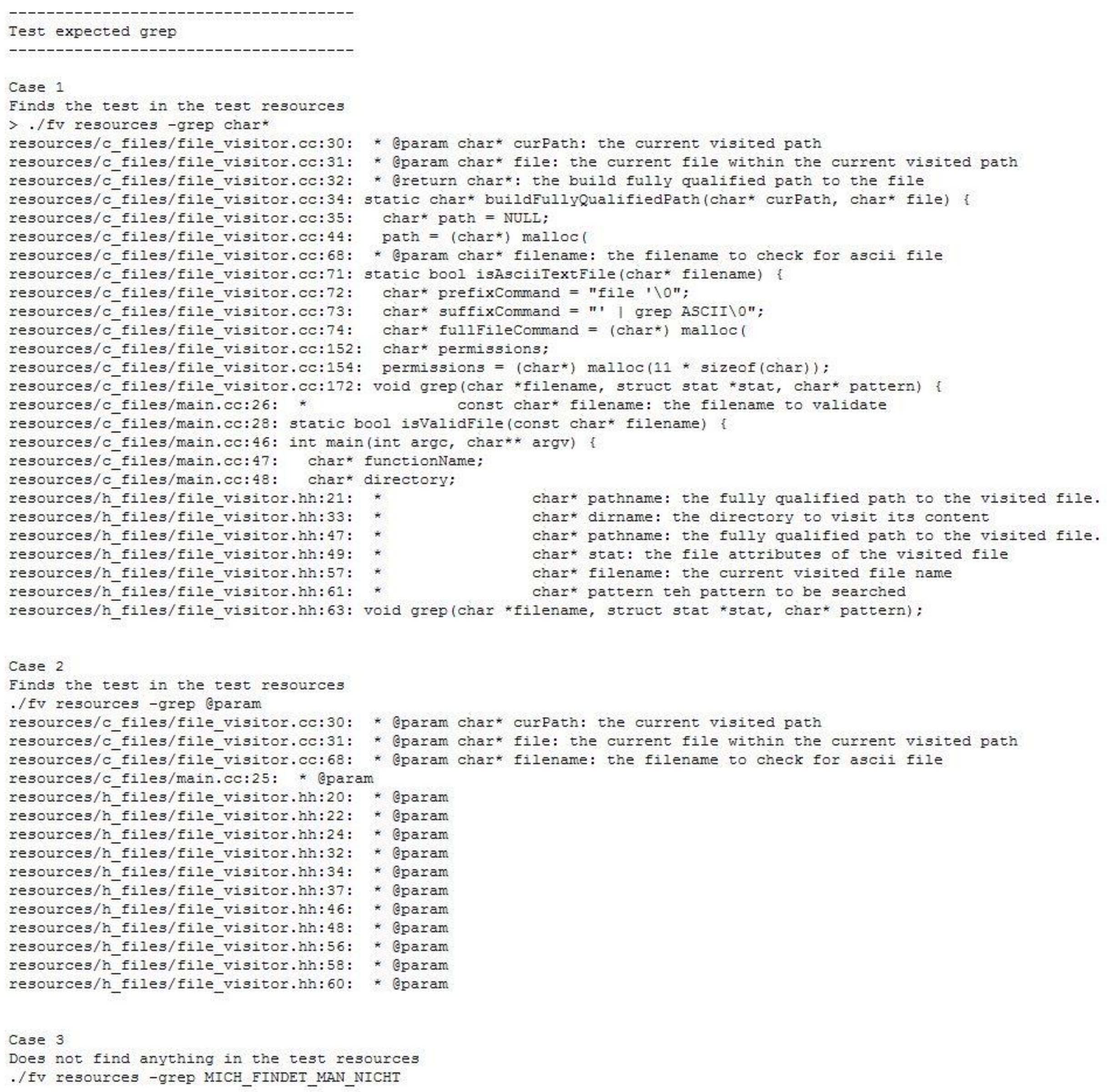
Folgend sind die Tests angeführt, welche das Verhalten bei der Angabe von ungültigen Argumenten beim Applikationsstart testen.



```
-------------------------------------
Test error handling
-------------------------------------


Case 1
No arguments given
> ./fv
At least the root directory and function must be given !!! E.g.: ./app.bin ./ -grep NULL

Case 2
Only directory given
> ./fv resources
At least the root directory and function must be given !!! E.g.: ./app.bin ./ -grep NULL

Case 3
grep without pattern
> ./fv resources -grep
Missing search pattern !!!

Case 4
invalid dir
> ./fv /invalidDir -grep
Given directory does not exist !!! dir='/invalidDir'

Case 5
invalid file
> ./fv /invalidDir/test.txt -grep
Given directory does not exist !!! dir='/invalidDir/test.txt'
```

1.2.2 Grep Funktion

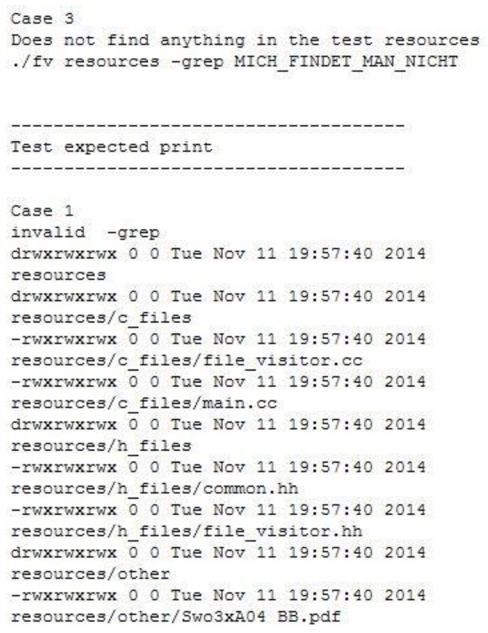
Folgend sind die Tests für die enthaltene Funktion grep angeführt, welche einen Text in den Dateien sucht.



```
-------------------------------------
Test expected grep
-------------------------------------

Case 1
Finds the test in the test resources
> ./fv resources -grep char*
resources/c_files/file_visitor.cc:30:  * @param char* curPath: the current visited path
resources/c_files/file_visitor.cc:31:  * @param char* file: the current file within the current visited path
resources/c_files/file_visitor.cc:32:  * @return char*: the build fully qualified path to the file
resources/c_files/file_visitor.cc:34: static char* buildFullyQualifiedPath(char* curPath, char* file) {
resources/c_files/file_visitor.cc:35:   char* path = NULL;
resources/c_files/file_visitor.cc:44:   path = (char*) malloc(
resources/c_files/file_visitor.cc:68:  * @param char* filename: the filename to check for ascii file
resources/c_files/file_visitor.cc:71: static bool isAsciiTextFile(char* filename) {
resources/c_files/file_visitor.cc:72:   char* prefixCommand = "file '\0";
resources/c_files/file_visitor.cc:73:   char* suffixCommand = "' | grep ASCII\0";
resources/c_files/file_visitor.cc:74:   char* fullFileCommand = (char*) malloc(
resources/c_files/file_visitor.cc:152:  char* permissions;
resources/c_files/file_visitor.cc:154:  permissions = (char*) malloc(11 * sizeof(char));
resources/c_files/file_visitor.cc:172: void grep(char *filename, struct stat *stat, char* pattern) {
resources/c_files/main.cc:26:  *                const char* filename: the filename to validate
resources/c_files/main.cc:28: static bool isValidFile(const char* filename) {
resources/c_files/main.cc:46: int main(int argc, char** argv) {
resources/c_files/main.cc:47:   char* functionName;
resources/c_files/main.cc:48:   char* directory;
resources/h_files/file_visitor.hh:21:  *                char* pathname: the fully qualified path to the visited file.
resources/h_files/file_visitor.hh:33:  *                char* dirname: the directory to visit its content
resources/h_files/file_visitor.hh:47:  *                char* pathname: the fully qualified path to the visited file.
resources/h_files/file_visitor.hh:49:  *                char* stat: the file attributes of the visited file
resources/h_files/file_visitor.hh:57:  *                char* filename: the current visited file name
resources/h_files/file_visitor.hh:61:  *                char* pattern teh pattern to be searched
resources/h_files/file_visitor.hh:63: void grep(char *filename, struct stat *stat, char* pattern);


Case 2
Finds the test in the test resources
./fv resources -grep @param
resources/c_files/file_visitor.cc:30:  * @param char* curPath: the current visited path
resources/c_files/file_visitor.cc:31:  * @param char* file: the current file within the current visited path
resources/c_files/file_visitor.cc:68:  * @param char* filename: the filename to check for ascii file
resources/c_files/main.cc:25:  * @param
resources/h_files/file_visitor.hh:20:  * @param
resources/h_files/file_visitor.hh:22:  * @param
resources/h_files/file_visitor.hh:24:  * @param
resources/h_files/file_visitor.hh:32:  * @param
resources/h_files/file_visitor.hh:34:  * @param
resources/h_files/file_visitor.hh:37:  * @param
resources/h_files/file_visitor.hh:46:  * @param
resources/h_files/file_visitor.hh:48:  * @param
resources/h_files/file_visitor.hh:56:  * @param
resources/h_files/file_visitor.hh:58:  * @param
resources/h_files/file_visitor.hh:60:  * @param


Case 3
Does not find anything in the test resources
./fv resources -grep MICH_FINDET_MAN_NICHT
```

1.2.3 Print Funktion

Folgend sind die Test angeführt, welche die Dateien und dessen Attribute auf die Konsole ausgibt.



```
Case 3
Does not find anything in the test resources
./fv resources -grep MICH_FINDET_MAN_NICHT


-------------------------------------
Test expected print
-------------------------------------

Case 1
invalid  -grep
drwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources
drwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/c_files
-rwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/c_files/file_visitor.cc
-rwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/c_files/main.cc
drwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/h_files
-rwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/h_files/common.hh
-rwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/h_files/file_visitor.hh
drwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/other
-rwxrwxrwx 0 0 Tue Nov 11 19:57:40 2014
resources/other/Swo3xA04 BB.pdf
```