Assignment 3

CSCI131

Name: James Marino

Username: jm617

Student Number: 4720994

Table Of Contents

1. Executive Summary

This report aims to simulate a Flat File Disk system. It has functions to simulate creation and deletion of files, and reading and writing values to the blocks of files. A defrag function is also implemented to clean up the disk space wise. Please see code and results below.

See comments for details on structure and outline of functionality of program.

2. Body

2.1 C Program

Header file for Disk Management below.
See comments for details.

Disk.h

```c
#ifndef DISK_H
#define DISK_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * Constants
 */
#define MAXFILES 64
#define DISKBLOCKS 512
#define NAMESIZE 32


/*
 * Structures
 */
// File System Errors
enum FileSystemErrors {
	NO_ERR,
	DIRECTORY_FULL,
	CREATE_FAIL,
	NON_EXISTENT_FILE,
	DUPLICATE_NAME,
	INVALID_BLOCK,
	ZERO_SIZE
};

// Directory Instance
struct DirEntry {
	// Filename of current file
	char FileName[NAMESIZE];
	// The start block number
	int Start;
	// Number of blocks allocated to file
	int Size;
	// Symbol representation
	char Symbol;
};
```

```
/*
 * Naming Conventions
 */
// File System Errors
typedef enum FileSystemErrors FileSysErrors;
// Directory Instance
typedef struct DirEntry DirectoryEntry;
// Bitmap
typedef int BitMap;
// Bool - 'cause we all wanna use C++
typedef int bool;


/*
 * Data
 */
// Directory List
static DirectoryEntry Directory[MAXFILES];
// Disk Bitmap - The actual HardDisk representation
static BitMap Disk[DISKBLOCKS];

// Stats
static int FilesCreated;
static int FilesDeleted;
static int FilesCompacted;

static int DirectoryEntries;
static int BlocksAllocated;
static int BlocksFree;

/*
 * Main Functions
 */
void initialiseFileSystem();

FileSysErrors createFile(const char* filename, int size);

FileSysErrors deleteFile(const char* filename);

FileSysErrors writeBlock(const char* filename, int block, int
value);

FileSysErrors readBlock(const char* filename, int block, int *
vp);

int compactFiles();

void displayDisk();

void showHistory();
```

```

/*
 * Helper Functions
 */
DirectoryEntry* getLowestDirPositionEntry(int last, bool
zeroPosition);

int findMemory(int fileSize);

FileSysErrors createFileErrorChecking(const char *filename, int
size);

FileSysErrors deleteFileErrorChecking(const char *fileName);

FileSysErrors readWriteBlockErrorChecking(const char *filename,
int block);

#endif
```

Main functions for Disk Management and helper functions below.
See comments for details.

Disk.c

```
#include "Disk.h"
#include <limits.h>

/*
 * ===================== Main Functions =====================
 */

/**
 * Format:
 * Initialises the file system
 * • Sets all blocks to 0 (ie empty)
 * • Set all Directory Entries
 * @return void
 */
void initialiseFileSystem()
{
     int count = 0;

     // Set all the disk blocks to 0
     for (count = 0; count < DISKBLOCKS; count++) {
          Disk[count] = 0;
     }

     // Set all Directory entries to Empty values
     for (count = 0; count < MAXFILES; count++) {

          // Set empty file name
          int filenameCount = 0;
          for (filenameCount = 0; filenameCount < NAMESIZE;
filenameCount++) {
               Directory[count].FileName[filenameCount] = '\0';
          }

          // Set empty size
          Directory[count].Size = 0;

          // Set empty record starting position
          Directory[count].Start = 0;

          // Set empty Symbol file representation (period)
          Directory[count].Symbol = '.';
     }

     // Stats
     FilesCreated = 0;
     FilesDeleted = 0;
     FilesCompacted = 0;
```

```
    DirectoryEntries = 1;
    BlocksAllocated = 0;
    BlocksFree = 0;
}

/**
 * Inserting files into the disk
 * @param const char* File Name
 * @param int Size of file to be inserted
 * @return FileSysErrors Errors for the function that have
occurred
 */
FileSysErrors createFile(const char* filename, int size)
{
    FileSysErrors errors;

    /* Error Checking Needed:
     * • Returns DIRECTORY_FULL if maxfiles used
     * • Returns DUPLICATE_NAME if there is already a file with
the name
     * • Returns ZERO_SIZE if size <= 0
     * • Returns CREATE_FAIL if unable to find space for the file
     */

    // 1. Check for simple errors before inserting
    errors = createFileErrorChecking(filename, size);

    if (errors != NO_ERR) {
        // Exit out
        return errors;
    } else {

            // 1. Check for full directory
            //if (DirectoryEntries > MAXFILES) {
            //    return DIRECTORY_FULL;
            //}

        // 2. Check for space and position to insert into HDD
        int position = findMemory(size);

        if (position < 0) {
            errors = CREATE_FAIL;
            return errors;
        } else {

            // 3. Insert into directory
            int count = 0;
            for (count = 0; count < MAXFILES; count++) {
                if (Directory[count].Size == 0) {

                    // Copy in data
                    strcpy(Directory[count].FileName,
filename);
                    Directory[count].Start = position;
                    Directory[count].Size = size;

                    break;
                }
            }

            // 4. Insert into Disk
            for (count = 0; count < size; count++) {
                // Mark positions as read
                Disk[position] = 1;

                // Increment the position
                position++;
            }

            // Update stats
            FilesCreated++;

            // All is good, return no error
            errors = NO_ERR;
            return errors;
        }

    }
}

/**
 * Deletes file from disk if available
 * @param const char* File Name
 * @return FileSysErrors appropriate errors
 */
FileSysErrors deleteFile(const char* filename)
{
    FileSysErrors errors;

    /* Error checking needed
     * returns NON_EXISTENT_FILE if filename is invalid
     */
    errors = deleteFileErrorChecking(filename);

    if (errors == NO_ERR) {

        // Temp vars
        DirectoryEntry tempDirEntry;

        // Get File properties from directory
        int counter = 0;
        for (counter = 0; counter < MAXFILES; counter++) {

            // Check for correct record
            if (strcmp(filename, Directory[counter].FileName) 
== 0) {
                tempDirEntry = Directory[counter];
                break;
            }
        }

        // Remove from Bit Map
        for (counter = 0; counter < tempDirEntry.Size; 
counter++) {

            // Format Section
            Disk[tempDirEntry.Start + counter] = 0;

        }

        // Remove from directory structure
        for (counter = 0; counter < MAXFILES; counter++) {

            // Check for correct record
            if (strcmp(filename, Directory[counter].FileName) 
== 0) {
                // Set empty file name
                int filenameCount = 0;
                for (filenameCount = 0; filenameCount < 
NAMESIZE; filenameCount++) {

Directory[counter].FileName[filenameCount] = '\0';
                }

                // Set empty size
                Directory[counter].Size = 0;

                // Set empty record starting position
                Directory[counter].Start = 0;

                // Set empty Symbol file representation 
(period)

                Directory[counter].Symbol = '.';

                break;
            }
        }

        // Update stats
        FilesDeleted++;

        return errors;

    } else {
        return errors;
    }
```

```
}

/**
 * Writes a single block value to the disk represented as an int
 * @param const char* File name
 * @param int block - position at which the block is inserted
 * @param int value - the actual value of what needs to be
inserted into the position
 * @return FileSysErrors Errors in execution
 */
FileSysErrors writeBlock(const char* filename, int block, int
value)
{
    FileSysErrors errors;

    /* Error checking needed
     * • return NON_EXISTENT_FILE if filename is invalid / not
found
     * • return INVALID_BLOCK if requested block is invalid (<0,
or >= size of file)
     */

    errors = readWriteBlockErrorChecking(filename, block);

    if (errors == NO_ERR) {

        // Temps
        int counter = 0;
        DirectoryEntry tempDirEntry;

        // Update block in file
        for (counter = 0; counter < MAXFILES; counter++) {
            // Compare all filenames
            if (strcmp(Directory[counter].FileName, filename)
== 0) {

                // Get the size of record
                tempDirEntry = Directory[counter];

                break;
            }
        }

        // Get the block location and store
        Disk[block + tempDirEntry.Start] = value;

        errors = NO_ERR;
        return errors;

    } else {
        return errors;
    }
```

```
}

/**
 * Read block - stores value read in integer address
 * @param const char * File Name
 * @param int block
 * @param int* address to value
 * @return FileSysErrors any errors
 */
FileSysErrors readBlock(const char* filename, int block, int *vp)
{
    FileSysErrors error;

    error = readWriteBlockErrorChecking(filename, block);

    if (error == NO_ERR) {

        // Temps
        int counter = 0;
        DirectoryEntry tempDirEntry;

        // Update block in file
        for (counter = 0; counter < MAXFILES; counter++) {
            // Compare all filenames
            if (strcmp(Directory[counter].FileName, filename)
== 0) {

                // Get the size of record
                tempDirEntry = Directory[counter];

                break;
            }
        }

        // Specify the address
        *vp = Disk[block + tempDirEntry.Start];

        error = NO_ERR;
        return error;

    } else {
        return error;
    }
}

/**
 * Defrag - groups files and defrags the disk
 * @return int the number of free blocks
 */
int compactFiles()
{
    // Global Counter
    int counter = 0;

    // Get current directory listing (start with lowest)
    DirectoryEntry *current = getLowestDirPositionEntry(0, 0);

    // Get the next lowest directory listing (next postition
above current dir listing)
    DirectoryEntry *next =
getLowestDirPositionEntry(current->Start, 0);

    // Loop until all records are done, there is nothing smaller
    while (next != NULL) {

        if ((current->Size + current->Start) == next->Start) {
            // Already compact, nothing we can do
        } else {

            // Get gap = (current dir (size + position) - next
dir (position))
            int gap = next->Start - (current->Size +
current->Start);
            int appart = gap;

            // 2. Update bit map

            for (counter = 0; counter < next->Size; counter++)
{
                Disk[next->Start - appart] =
Disk[next->Start + counter];
                Disk[next->Start + counter] = 0;
                appart--;
            }

            // Work on next cause thats what we are bringing
closer
            // 1. File Listing related fix up
            next->Start = next->Start - gap;

            // loop over (next dir listing file size) {

            // bitmap[next-file.position - gap] =
bitmap[next-file.position]
            // bitmap[next-file.position] = 0

        }

        current = next;
        next = getLowestDirPositionEntry(current->Start, 1);

    }
    // Get number of free blocks
    int freeBlocks = 0;
    for (counter = 0; counter < DISKBLOCKS; counter++) {

        if (Disk[counter] == 0) {
            freeBlocks++;
        }
    }

    // Stats
    FilesCompacted++;

    return freeBlocks;
}

/**
 * Display disk - show directory contents and provide some form of
map
 * identifying the mapping of files to disk blocks
 */
void displayDisk()
{
    // Setup final array to be printed
    char ASCIIDisk[DISKBLOCKS];

    // Counter and Initialisation
    int counter = 0;
    // Set all blocks to have period
    for (counter = 0; counter < DISKBLOCKS; counter++) {
        ASCIIDisk[counter] = '.';
    }

    // Setup Symbol counter
    char symbol = 65;

    // Iterate through directory
    for (counter = 0; counter < MAXFILES; counter++) {

        // Symbol counter
        int symbolCounter = 0;
        int position = Directory[counter].Start;

        // Go through the whole size of the file
        for (symbolCounter = 0; symbolCounter <
Directory[counter].Size; symbolCounter++) {
            // Set starting postion
            ASCIIDisk[position] = symbol;

            // Update position
            position++;
        }
        // Set the symbol used
        Directory[counter].Symbol = symbol;

        // Get a new symbol
        // Check bounds - just incase...
        if (symbol >= 91) {
            symbol = 97;
        } else if (symbol >= 123) {
            symbol = 65;
        } else {
            symbol++;
        }

    }

    // Print out the list of files
    printf("Directory Listing:  Filename:   Start:  Size:
(Symbol): \n");
    // Print out directory ls
    for (counter = 0; counter < MAXFILES; counter++) {
        // No Blank records
        if (Directory[counter].Size != 0) {
            printf("%d             %s     %d     %d
%c     \n",
                  counter+1,
                  Directory[counter].FileName,
                  Directory[counter].Start,
                  Directory[counter].Size,
                  Directory[counter].Symbol
                  );
        }
    }

    printf("\n");
    // Print out the display of files
    for (counter = 0; counter < DISKBLOCKS; counter++) {

        // Print in 64 Block lines
        if ((counter % 64) == 0) {
            printf("\n");
        }

        // Out the characters
        printf("%c", ASCIIDisk[counter]);
    }

    printf("\n\n");

}

// show history
// number of files created, deleted, number of entries in
```

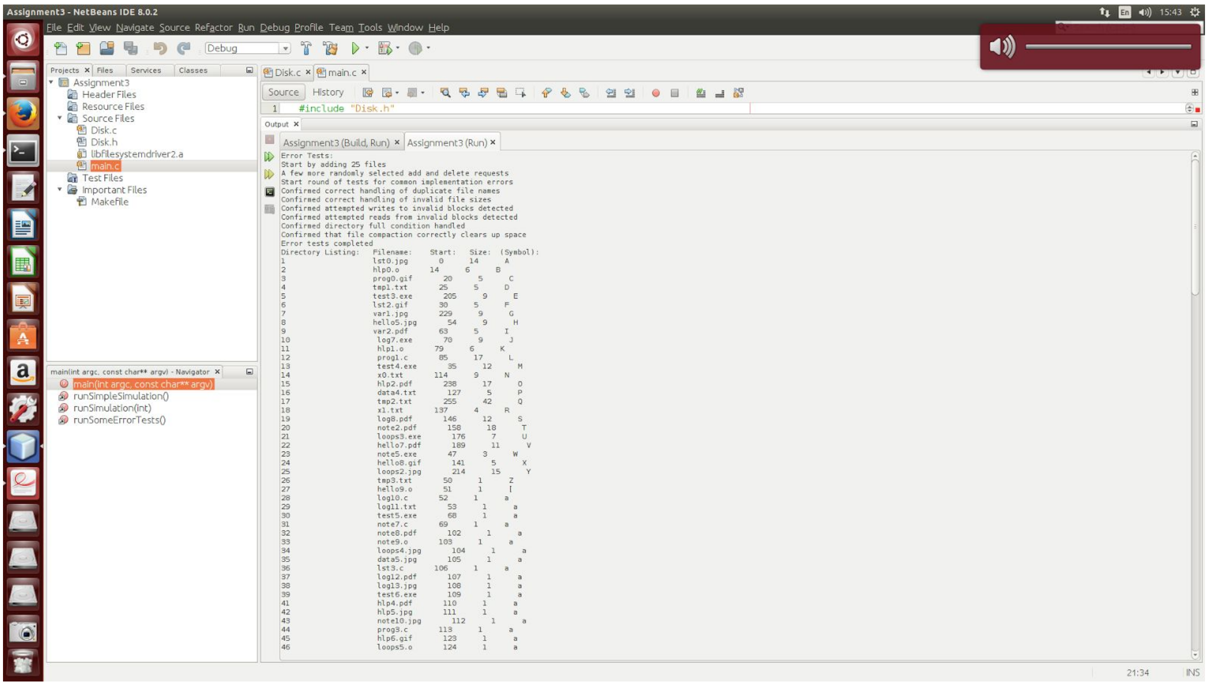
```
directory,
// number of blocks still free, number of compactions performed
void showHistory()
{

    int counter = 0;
    for (counter = 0; counter < DISKBLOCKS; counter++) {

        if (Disk[counter] == 0) {
            BlocksFree++;
        }
    }

    BlocksAllocated = DISKBLOCKS - BlocksFree;

    for (counter = 0; counter < MAXFILES; counter++) {

        if (Directory[counter].Size != 0) {
            DirectoryEntries++;
        }

    }

    printf("\n");
    printf("Number of file create operations        : %d\n",
FilesCreated);
    printf("Number of file delete operations        : %d\n",
FilesDeleted);
    printf("Number of file compaction operations    : %d\n",
FilesCompacted);
    printf("\n\n");
    printf("Current number of directory entries     : %d\n",
DirectoryEntries);
    printf("Current number of disk blocks allocated : %d\n",
BlocksAllocated);
    printf("Current number of disk blocks free      : %d\n",
BlocksFree);
    printf("\n");
}

/*
 * ===================== Helper Functions =====================
 */

/**
 * Gets the next lowest directory position above the last position
 * @param int Last position to be used
 * @return Directory Entry details
 */
DirectoryEntry* getLowestDirPositionEntry(int last, bool
zeroPosition)
{
    int smallest = INT_MAX;
    DirectoryEntry* lowest = NULL;
    int counter = 0;

    // Initial
    if ((last == 0) && (zeroPosition == 1)) {

        // Check if lowest is 0
        for (counter = 0; counter < MAXFILES; counter++) {

            if (Directory[counter].Start == last) {

                // It is found, return the directory listing
                return &Directory[counter];

            }
        }
    }

    bool nothingGreaterThanLast = 0;

    for (counter = 0; counter < MAXFILES; counter++) {



        if (Directory[counter].Start > last) {

            // Set flag that there is something greater
            nothingGreaterThanLast = 1;

            // if the value we are testing is smaller than the
smallest
            if (Directory[counter].Start < smallest) {

                // Set that value as the smallest
                smallest = Directory[counter].Start;
                // Set the pointer
                lowest = &Directory[counter];
            }

        }

    }

    if (nothingGreaterThanLast == 0) {

        // Nothing more to be found
        return NULL;
    } else {
        return lowest;
    }
```

```
}

/**
 * First Fit - Find Memory:
 * Start at begginig of HDD and search through for
 * an appropriate sequence of fee blocks = to the requested size
 * @param int File size
 * @return int Block starting location (-1 if error)
 */
int findMemory(int fileSize)
{

     // Setup counters
     int diskPosition = 0;
     int sequencePosition = 0;

     // Cycle through the whole disk
     for (diskPosition = 0; diskPosition < DISKBLOCKS;
diskPosition++) {

          // Check at postion _diskPosition for a valid sequence
          for (sequencePosition = 0; sequencePosition <=
fileSize; sequencePosition++) {

               // Get the current position of array while in the
loop
               // Check the next element in the array
               int currentPositition = diskPosition +
sequencePosition;

               // Check if we are outside the range of position
               if (currentPositition > DISKBLOCKS) {

                    // return error
                    return -1;
               }

               if (Disk[currentPositition] != 0) {
                    // Non empty block found, sequence is
interuppted
                    // fast forward main position
                    diskPosition = diskPosition +
sequencePosition;

                    // break out of loop, nothing else to find
                    break;
               }

               // Only when gone though loop with no break set
the flag
               if (sequencePosition == fileSize) {
                return diskPosition;

            }

        }

    }

    // Nothing found, error out
    return -1;

}

/**
 * Check errors before writing block
 * @param const char * File Name
 * @param int block to be inserted
 * @param int value to be inserted
 * @return FileSysErrors
 * • return NON_EXISTENT_FILE if filename is invalid / not found
 * • return INVALID_BLOCK if requested block is invalid (<0, or >=
size of file)
 */
FileSysErrors readWriteBlockErrorChecking(const char *filename,
int block)
{
    FileSysErrors error;

    int counter = 0;
    int tempSize = 0;

    // 1. Check if filename is found
    {
        bool nameFound = 0;

        for (counter = 0; counter < MAXFILES; counter++) {
            // Compare all filenames
            if (strcmp(Directory[counter].FileName, filename)
== 0) {

                // Get the size of record
                tempSize = Directory[counter].Size;

                // Strings are equal
                nameFound = 1;
                break;
            }
        }

        if (nameFound == 0) {
            error = NON_EXISTENT_FILE;
            return error;
        }
    }

    // 2. Invalid Filename
    {
        if (strlen(filename) <= 0) {
            error = NON_EXISTENT_FILE;
            return error;
        }
    }

    // 3. Invalid Blocks
    {
        // INVALID_BLOCK if requested block is invalid (<0, or
>= size of file)
        if ((block < 0) || (block >= tempSize)) {
            error = INVALID_BLOCK;
            return error;
        }
    }

    // Say no errors for now if nothing found
    error = NO_ERR;
    return error;
}

/**
 * Check simple errors before creating a file
 * @param const char* File Name
 * @param int Size of the file
 * @return FileSysErrors Errors found so far
 *    • Returns DIRECTORY_FULL if maxfiles used
 *    • Returns DUPLICATE_NAME if there is already a file with the
name
 *    • Returns ZERO_SIZE if size <= 0
 */
FileSysErrors createFileErrorChecking(const char *filename, int
size)
{
    // Local Vars
    FileSysErrors error;
    int counter = 0;

    // 1. Check directory is full
    {

        bool freeSpaceFound = 0;

        for (counter = 0; counter < MAXFILES; counter++) {
            if (Directory[counter].Size == 0) {
                // There is free space
                freeSpaceFound = 1;
                break;
            }
        }

        if (freeSpaceFound == 0) {
            error = DIRECTORY_FULL;
            return error;
        }

    }

    // 2. Check duplicate file names
    {
        for (counter = 0; counter < MAXFILES; counter++) {
            // Compare all filenames
            if (strcmp(Directory[counter].FileName, filename)
== 0) {
                // Strings are equal
                error = DUPLICATE_NAME;
                return error;
            }
        }
    }

    // 3. Check for 0 Size
    {
        if (size <= 0) {
            error = ZERO_SIZE;
            return error;
        }
    }

    // Say no errors for now if nothing found
    error = NO_ERR;
    return error;
}

/**
 * Check simple errors before performing directory operations IO
 * Relating To:
 *    • DIRECTORY_FULL
 *    • DUPLICATE_NAME
 *    • ZERO_SIZE
 *    • NON_EXISTENT_FILE
 *    • INVALID_BLOCK
 * @param const char * File Name
 * @return FileSysErrors File system errors
 */
FileSysErrors deleteFileErrorChecking(const char *fileName)
{
    // Local Vars
    FileSysErrors error;
    int counter = 0;

    // 1. Check if is not existing filename
    {
        bool flag = 0;

        for (counter = 0; counter < MAXFILES; counter++) {
            // Compare all filenames
            if (strcmp(Directory[counter].FileName, fileName)
== 0) {
                // Strings are equal
                // Set flag a match is found
                flag = 1;

                // Exit out
                break;
            }
        }

        // Test for non existance
        if (flag != 1) {
            error = NON_EXISTENT_FILE;
            return error;
        }
    }

    // 2. Check Invalid filename
    {
        if (strlen(fileName) <= 0) {
            error = NON_EXISTENT_FILE;
            return error;
        }
    }

    // Say no errors for now
    error = NO_ERR;
    return error;
}
```

2.3 Results

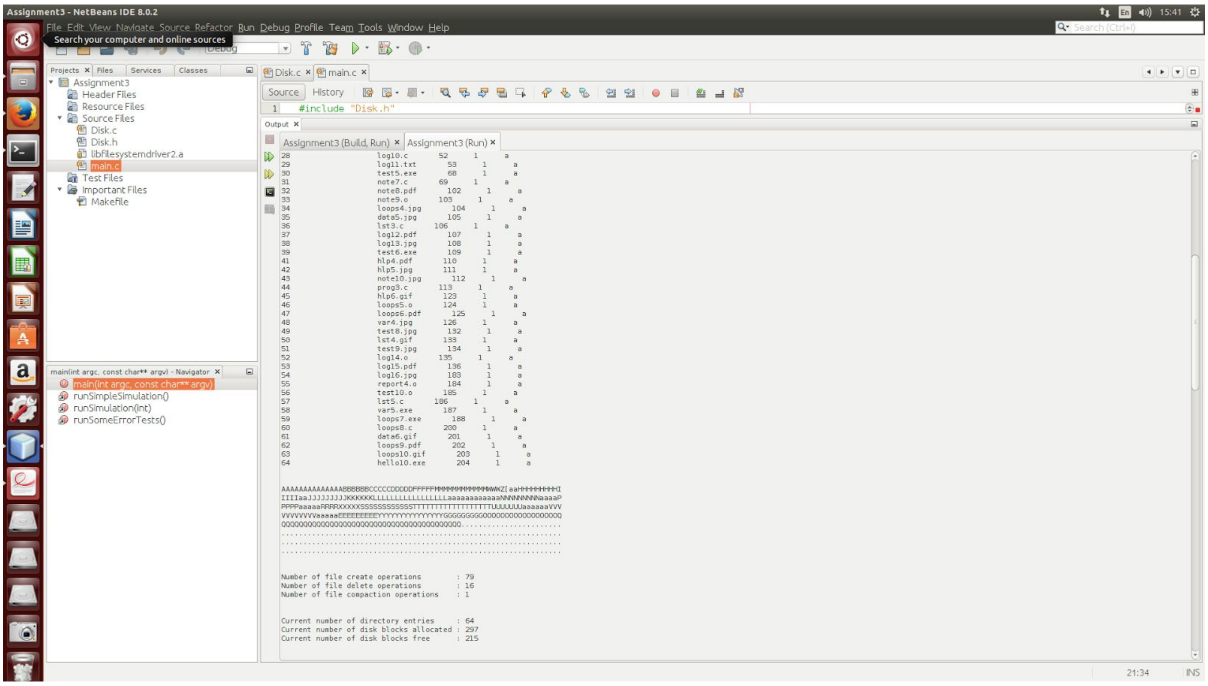
Screenshots:

Error Testing:



```
#include "Disk.h"
```

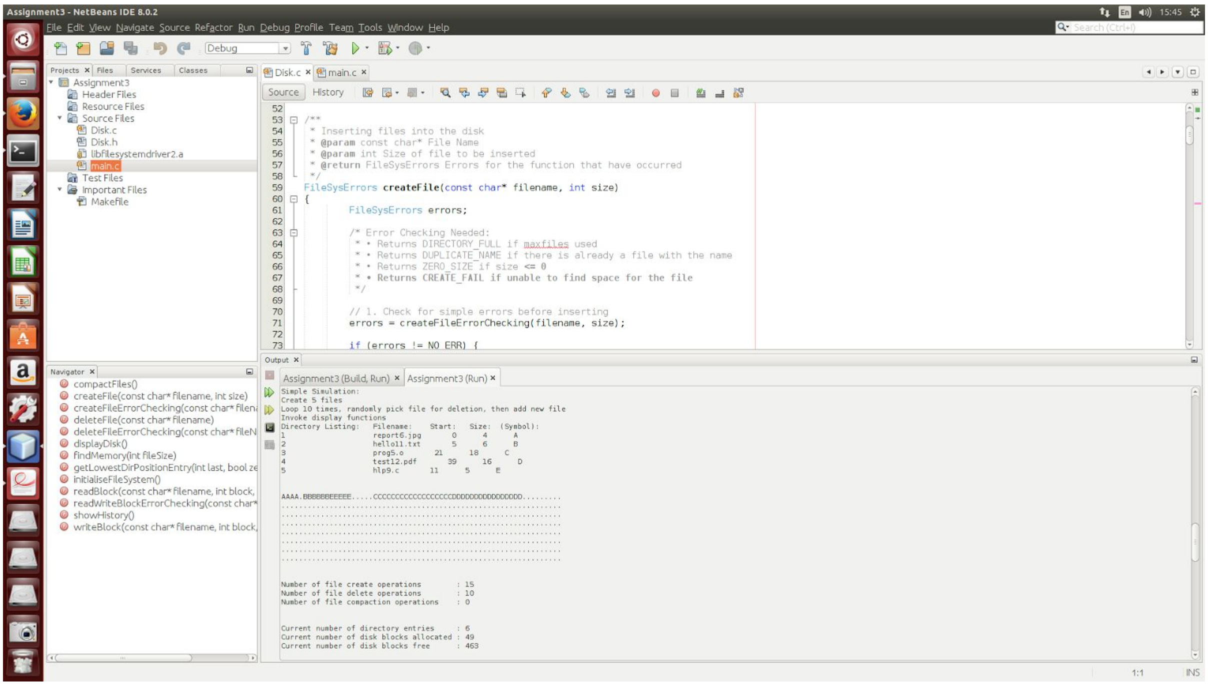
```
Error Tests:
Start by adding 25 files
A few more randomly selected add and delete requests
Start round of tests for common implementation errors
Confirmed correct handling of duplicate file names
Confirmed correct handling of invalid file sizes
Confirmed attempted writes to invalid blocks detected
Confirmed attempted reads from invalid blocks detected
Confirmed directory full condition handled
Confirmed that file compaction correctly clears up space
Error tests completed
Directory Listing:   Filename:    Start:   Size:  (Symbol):
1                    lst0.jpg       0      14       A
2                    hlp0.o      14      6      B
3                    prog0.gif      20      5       C
4                    tmp1.txt      25      5      D
5                    test3.exe      205      9      E
6                    lst2.gif      30      5      F
7                    var1.jpg      229      9      G
8                    hello5.jpg      54      9      H
9                    var2.pdf      63      5      I
10                    log7.exe      70      9      J
11                    hlp1.o      79      6      K
12                    prog1.c      85      17      L
13                    test4.exe      35      12      M
14                    x0.txt      114      9      N
15                    hlp2.pdf      238      17      O
16                    data4.txt      127      5      P
17                    tmp2.txt      255      42      Q
18                    x1.txt      137      4      R
19                    log8.pdf      146      12      S
20                    note2.pdf      158      18      T
21                    loops3.exe      176      7      U
22                    hello7.pdf      189      11      V
23                    note5.exe      47      3      W
24                    hello8.gif      141      5      X
25                    loops2.jpg      214      15      Y
26                    tmp3.txt      50      1      Z
27                    hello9.o      51      1      [
28                    log10.c      52      1      a
29                    log11.txt      53      1      a
30                    test5.exe      68      1      a
31                    note7.c      69      1      a
32                    note8.pdf      102      1      a
33                    note9.o      103      1      a
34                    loops4.jpg      104      1      a
35                    data5.jpg      105      1      a
36                    lst3.c      106      1      a
37                    log12.pdf      107      1      a
38                    log13.jpg      108      1      a
39                    test6.exe      109      1      a
41                    hlp4.pdf      110      1      a
42                    hlp5.jpg      111      1      a
43                    note10.jpg      112      1      a
44                    prog3.c      113      1      a
45                    hlp6.gif      123      1      a
46                    loops5.o      124      1      a
```



```
#include "Disk.h"
```

```
28                    log10.c      52      1      a
29                    log11.txt      53      1      a
30                    test5.exe      68      1      a
31                    note7.c      69      1      a
32                    note8.pdf      102      1      a
33                    note9.o      103      1      a
34                    loops4.jpg      104      1      a
35                    data5.jpg      105      1      a
36                    lst3.c      106      1      a
37                    log12.pdf      107      1      a
38                    log13.jpg      108      1      a
39                    test6.exe      109      1      a
41                    hlp4.pdf      110      1      a
42                    hlp5.jpg      111      1      a
43                    note10.jpg      112      1      a
44                    prog3.c      113      1      a
45                    hlp6.gif      123      1      a
46                    loops5.o      124      1      a
47                    loops6.pdf      125      1      a
48                    var4.jpg      126      1      a
49                    test8.jpg      132      1      a
50                    lst4.gif      133      1      a
51                    test9.jpg      134      1      a
52                    log14.o      135      1      a
53                    log15.pdf      136      1      a
54                    log16.jpg      183      1      a
55                    report4.o      184      1      a
56                    test10.o      185      1      a
57                    lst5.c      186      1      a
58                    var5.exe      187      1      a
59                    loops7.exe      188      1      a
60                    loops8.c      200      1      a
61                    data6.gif      201      1      a
62                    loops9.pdf      202      1      a
63                    loops10.gif      203      1      a
64                    hello10.exe      204      1      a


AAAAAAAAAAAAAABBBBBBCCCCCDDDDDFFFFFMMMMMMMMMMMMWWWZ[aaHHHHHHHHHI
IIIIaaJJJJJJJJJKKKKKKLLLLLLLLLLLLLLLLLaaaaaaaaaaaaNNNNNNNNNaaaaP
PPPPaaaaaRRRRXXXXXSSSSSSSSSSSSTTTTTTTTTTTTTTTTTTUUUUUUUaaaaaaVVV
VVVVVVVVaaaaaEEEEEEEEEYYYYYYYYYYYYYYYGGGGGGGGGOOOOOOOOOOOOOOOOOQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ.......................
................................................................
................................................................
................................................................


Number of file create operations        : 79
Number of file delete operations        : 16
Number of file compaction operations    : 1


Current number of directory entries     : 64
Current number of disk blocks allocated : 297
Current number of disk blocks free      : 215
```

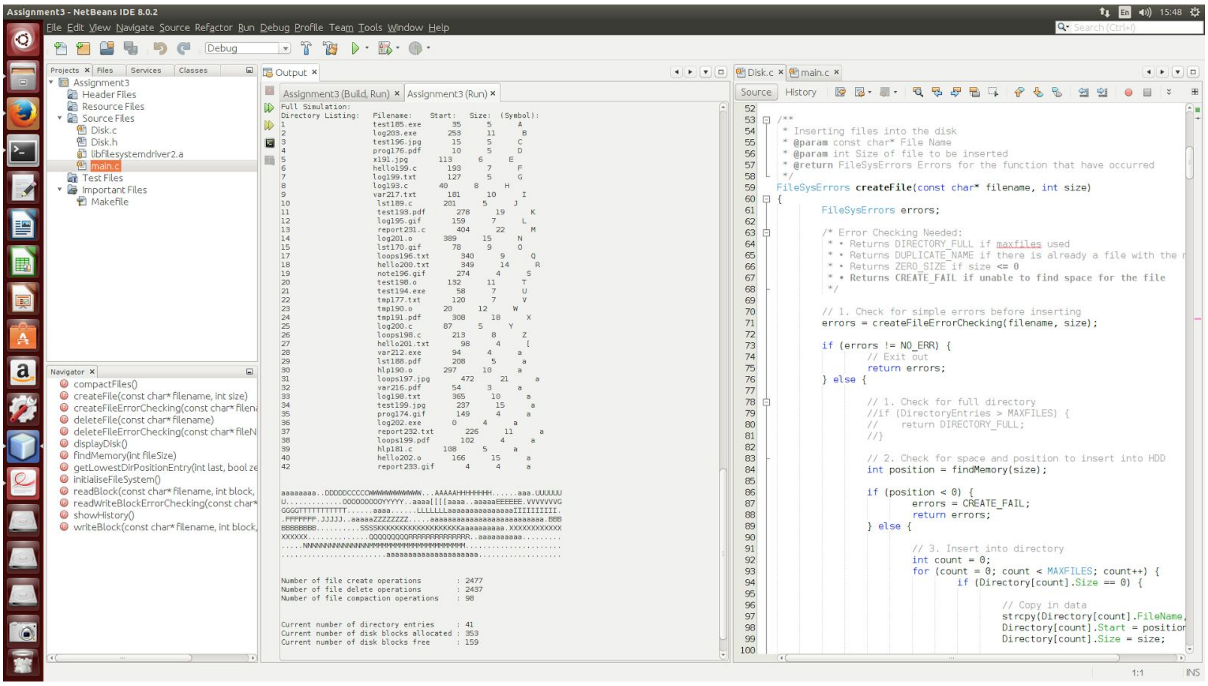
Simple Simulation:


Assignment3 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Debug
Projects Files Services Classes
Assignment3
Header Files
Resource Files
Source Files
Disk.c
Disk.h
libfilesystemdriver2.a
main.c
Test Files
Important Files
Makefile
Navigator
compactFiles()
createFile(const char* filename, int size)
createFileErrorChecking(const char* filen
deleteFile(const char* filename)
deleteFileErrorChecking(const char* fileN
displayDisk()
findMemory(int fileSize)
getLowestDirPositionEntry(int last, bool ze
initialiseFileSystem()
readBlock(const char* filename, int block,
readWriteBlockErrorChecking(const char*
showHistory()
writeBlock(const char* filename, int block,
Disk.c main.c
Source History
Output
Assignment3 (Build, Run) Assignment3 (Run)
15:45
1:1 INS

```c
/**
 * Inserting files into the disk
 * @param const char* File Name
 * @param int Size of file to be inserted
 * @return FileSysErrors Errors for the function that have occurred
 */
FileSysErrors createFile(const char* filename, int size)
{
        FileSysErrors errors;

        /* Error Checking Needed:
         * • Returns DIRECTORY_FULL if maxfiles used
         * • Returns DUPLICATE_NAME if there is already a file with the name
         * • Returns ZERO_SIZE if size <= 0
         * • Returns CREATE_FAIL if unable to find space for the file
         */

        // 1. Check for simple errors before inserting
        errors = createFileErrorChecking(filename, size);

        if (errors != NO_ERR) {
```

```
Simple Simulation:
Create 5 files
Loop 10 times, randomly pick file for deletion, then add new file
Invoke display functions
Directory Listing:   Filename:    Start:   Size:  (Symbol):
1                    report6.jpg       0      4      A
2                    hello11.txt       5      6      B
3                    prog5.o       21      18      C
4                    test12.pdf       39      16      D
5                    hlp9.c       11      5      E


AAAA.BBBBBBEEEEE.....CCCCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDD.........
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................


Number of file create operations        : 15
Number of file delete operations        : 10
Number of file compaction operations    : 0


Current number of directory entries     : 6
Current number of disk blocks allocated : 49
Current number of disk blocks free      : 463
```

Full Simulation:


Assignment3 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Debug
Projects Files Services Classes
Assignment3
Header Files
Resource Files
Source Files
Disk.c
Disk.h
libfilesystemdriver2.a
main.c
Test Files
Important Files
Makefile
Navigator
compactFiles()
createFile(const char* filename, int size)
createFileErrorChecking(const char* filen
deleteFile(const char* filename)
deleteFileErrorChecking(const char* fileN
displayDisk()
findMemory(int fileSize)
getLowestDirPositionEntry(int last, bool ze
initialiseFileSystem()
readBlock(const char* filename, int block,
readWriteBlockErrorChecking(const char*
showHistory()
writeBlock(const char* filename, int block,
Output
Assignment3 (Build, Run) Assignment3 (Run)
Disk.c main.c
Source History
15:48
Search (Ctrl+I)
1:1 INS

```
Full Simulation:
Directory Listing:   Filename:    Start:   Size:  (Symbol):
1                    test185.exe       35      5      A
2                    log203.exe       253      11      B
3                    test196.jpg       15      5      C
4                    prog176.pdf       10      5      D
5                    x191.jpg       113      6      E
6                    hello199.c       193      7      F
7                    log199.txt       127      5      G
8                    log193.c       40      8      H
9                    var217.txt       181      10      I
10                    lst189.c       201      5      J
11                    test193.pdf       278      19      K
12                    log195.gif       159      7      L
13                    report231.c       404      22      M
14                    log201.o       389      15      N
15                    lst170.gif       78      9      O
17                    loops196.txt       340      9      Q
18                    hello200.txt       349      14      R
19                    note196.gif       274      4      S
20                    test198.o       132      11      T
21                    test194.exe       58      7      U
22                    tmp177.txt       120      7      V
23                    tmp190.o       20      12      W
24                    tmp191.pdf       308      18      X
25                    log200.c       87      5      Y
26                    loops198.c       213      8      Z
27                    hello201.txt       98      4      [
28                    var212.exe       94      4      a
29                    lst188.pdf       208      5      a
30                    hlp190.o       297      10      a
31                    loops197.jpg       472      21      a
32                    var216.pdf       54      3      a
33                    log198.txt       365      10      a
34                    test199.jpg       237      15      a
35                    prog174.gif       149      4      a
36                    log202.exe       0      4      a
37                    report232.txt       226      11      a
38                    loops199.pdf       102      4      a
39                    hlp181.c       108      5      a
40                    hello202.o       166      15      a
42                    report233.gif       4      4      a

aaaaaaaa..DDDDDCCCCCWWWWWWWWWWWW...AAAAAHHHHHHHH......aaa.UUUUUU
U.............OOOOOOOOOYYYYY..aaaa[[[[aaaa..aaaaaEEEEEE.VVVVVVVG
GGGGTTTTTTTTTTT......aaaa......LLLLLLLaaaaaaaaaaaaaaaIIIIIIIIII.
.FFFFFFF.JJJJJ..aaaaaZZZZZZZZ....aaaaaaaaaaaaaaaaaaaaaaaaaaa.BBB
BBBBBBBB..........SSSSKKKKKKKKKKKKKKKKKKKaaaaaaaaaa.XXXXXXXXXXXX
XXXXXX..............QQQQQQQQQRRRRRRRRRRRRRR..aaaaaaaaaa.........
.....NNNNNNNNNNNNNNNMMMMMMMMMMMMMMMMMMMMMM......................
........................aaaaaaaaaaaaaaaaaaaaa...................


Number of file create operations        : 2477
Number of file delete operations        : 2437
Number of file compaction operations    : 98


Current number of directory entries     : 41
Current number of disk blocks allocated : 353
Current number of disk blocks free      : 159
```

```c
/**
 * Inserting files into the disk
 * @param const char* File Name
 * @param int Size of file to be inserted
 * @return FileSysErrors Errors for the function that have occurred
 */
FileSysErrors createFile(const char* filename, int size)
{
        FileSysErrors errors;

        /* Error Checking Needed:
         * • Returns DIRECTORY_FULL if maxfiles used
         * • Returns DUPLICATE_NAME if there is already a file with the r
         * • Returns ZERO_SIZE if size <= 0
         * • Returns CREATE_FAIL if unable to find space for the file
         */

        // 1. Check for simple errors before inserting
        errors = createFileErrorChecking(filename, size);

        if (errors != NO_ERR) {
                // Exit out
                return errors;
        } else {

                // 1. Check for full directory
                //if (DirectoryEntries > MAXFILES) {
                //    return DIRECTORY_FULL;
                //}

                // 2. Check for space and position to insert into HDD
                int position = findMemory(size);

                if (position < 0) {
                        errors = CREATE_FAIL;
                        return errors;
                } else {

                        // 3. Insert into directory
                        int count = 0;
                        for (count = 0; count < MAXFILES; count++) {
                                if (Directory[count].Size == 0) {

                                        // Copy in data
                                        strcpy(Directory[count].FileName,
                                        Directory[count].Start = positior
                                        Directory[count].Size = size;
```

3. Conclusion

As we can see the program runs as expected and provides the appropriate output in creating, deleting and compacting files.