

Assignment 3

CSCI131

Name: James Marino

Username: jm617

Student Number: 4720994

Table Of Contents

1. Executive Summary
2. Body
 - 2.1. C Program
 - 2.2. Results
3. Conclusion

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

```
#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 position
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 ASCIIIDisk[DISKBLOCKS];

    // Counter and Initialisation
    int counter = 0;
    // Set all blocks to have period
    for (counter = 0; counter < DISKBLOCKS; counter++) {
        ASCIIIDisk[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 position
            ASCIIIDisk[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", ASCIIIDisk[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 free 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 currentPosition = diskPosition +
sequencePosition;

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

                // return error
                return -1;
            }

            if (Disk[currentPosition] != 0) {
                // Non empty block found, sequence is
interrupted
                // 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 existence
    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

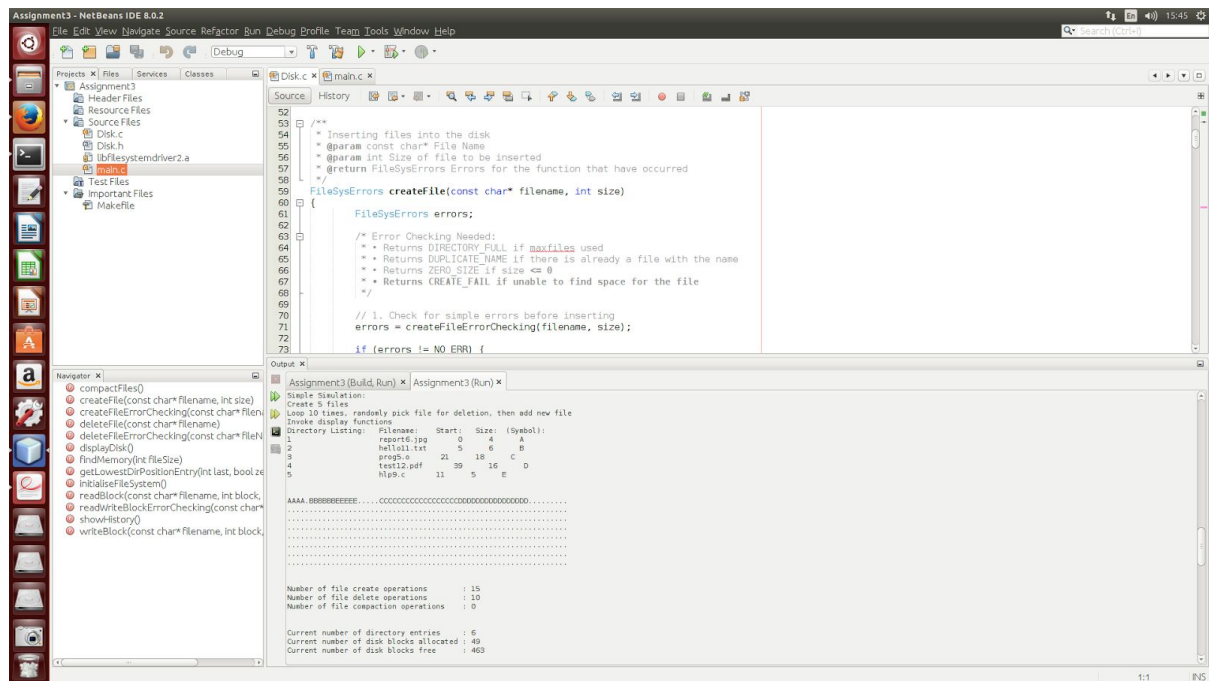
Screenshots:

Error Testing:

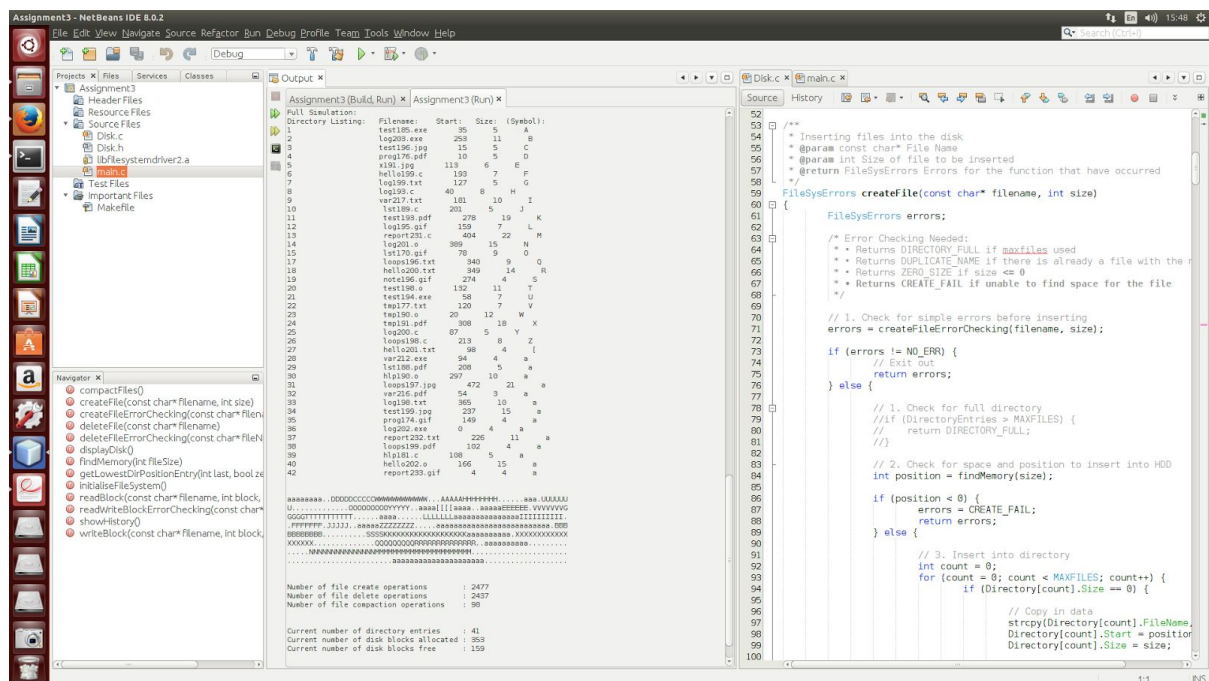
```

Assignment3 (Build, Run) | Assignment3 (Run)
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:
File Name:      Start:   Size: (Symbol):
1  log10.jpg      0      14      A
2  log11.jpg      14     16      B
3  log12.jpg      25     17      C
4  log13.jpg      30     18      D
5  log14.jpg      35     19      E
6  log15.jpg      40     20      F
7  log16.jpg      45     21      G
8  log17.jpg      50     22      H
9  log18.jpg      55     23      I
10 log19.jpg      60     24      J
11 log20.jpg      65     25      K
12 log21.jpg      70     26      L
13 log22.jpg      75     27      M
14 log23.jpg      80     28      N
15 log24.jpg      85     29      O
16 log25.jpg      90     30      P
17 log26.jpg      95     31      Q
18 log27.jpg      100    32      R
19 log28.jpg      105    33      S
20 log29.jpg      110    34      T
21 log30.jpg      115    35      U
22 log31.jpg      120    36      V
23 log32.jpg      125    37      W
24 log33.jpg      130    38      X
25 log34.jpg      135    39      Y
26 log35.jpg      140    40      Z
27 log36.jpg      145    41      a
28 log37.jpg      150    42      b
29 log38.jpg      155    43      c
30 log39.jpg      160    44      d
31 log40.jpg      165    45      e
32 log41.jpg      170    46      f
33 log42.jpg      175    47      g
34 log43.jpg      180    48      h
35 log44.jpg      185    49      i
36 log45.jpg      190    50      j
37 log46.jpg      195    51      k
38 log47.jpg      200    52      l
39 log48.jpg      205    53      m
40 log49.jpg      210    54      n
41 log50.jpg      215    55      o
42 log51.jpg      220    56      p
43 log52.jpg      225    57      q
44 log53.jpg      230    58      r
45 log54.jpg      235    59      s
46 log55.jpg      240    60      t
47 log56.jpg      245    61      u
48 log57.jpg      250    62      v
49 log58.jpg      255    63      w
50 log59.jpg      260    64      x
51 log60.jpg      265    65      y
52 log61.jpg      270    66      z
53 log62.jpg      275    67      A
54 log63.jpg      280    68      B
55 log64.jpg      285    69      C
56 log65.jpg      290    70      D
57 log66.jpg      295    71      E
58 log67.jpg      300    72      F
59 log68.jpg      305    73      G
60 log69.jpg      310    74      H
61 log70.jpg      315    75      I
62 log71.jpg      320    76      J
63 log72.jpg      325    77      K
64 log73.jpg      330    78      L
65 log74.jpg      335    79      M
66 log75.jpg      340    80      N
67 log76.jpg      345    81      O
68 log77.jpg      350    82      P
69 log78.jpg      355    83      Q
70 log79.jpg      360    84      R
71 log80.jpg      365    85      S
72 log81.jpg      370    86      T
73 log82.jpg      375    87      U
74 log83.jpg      380    88      V
75 log84.jpg      385    89      W
76 log85.jpg      390    90      X
77 log86.jpg      395    91      Y
78 log87.jpg      400    92      Z
79 log88.jpg      405    93      a
80 log89.jpg      410    94      b
81 log90.jpg      415    95      c
82 log91.jpg      420    96      d
83 log92.jpg      425    97      e
84 log93.jpg      430    98      f
85 log94.jpg      435    99      g
86 log95.jpg      440   100      h
87 log96.jpg      445   101      i
88 log97.jpg      450   102      j
89 log98.jpg      455   103      k
90 log99.jpg      460   104      l
91 log100.jpg     465   105      m
92 log101.jpg     470   106      n
93 log102.jpg     475   107      o
94 log103.jpg     480   108      p
95 log104.jpg     485   109      q
96 log105.jpg     490   110      r
97 log106.jpg     495   111      s
98 log107.jpg     500   112      t
99 log108.jpg     505   113      u
100 log109.jpg    510   114      v
101 log110.jpg    515   115      w
102 log111.jpg    520   116      x
103 log112.jpg    525   117      y
104 log113.jpg    530   118      z
105 log114.jpg    535   119      A
106 log115.jpg    540   120      B
107 log116.jpg    545   121      C
108 log117.jpg    550   122      D
109 log118.jpg    555   123      E
110 log119.jpg    560   124      F
111 log120.jpg    565   125      G
112 log121.jpg    570   126      H
113 log122.jpg    575   127      I
114 log123.jpg    580   128      J
115 log124.jpg    585   129      K
116 log125.jpg    590   130      L
117 log126.jpg    595   131      M
118 log127.jpg    600   132      N
119 log128.jpg    605   133      O
120 log129.jpg    610   134      P
121 log130.jpg    615   135      Q
122 log131.jpg    620   136      R
123 log132.jpg    625   137      S
124 log133.jpg    630   138      T
125 log134.jpg    635   139      U
126 log135.jpg    640   140      V
127 log136.jpg    645   141      W
128 log137.jpg    650   142      X
129 log138.jpg    655   143      Y
130 log139.jpg    660   144      Z
131 log140.jpg    665   145      a
132 log141.jpg    670   146      b
133 log142.jpg    675   147      c
134 log143.jpg    680   148      d
135 log144.jpg    685   149      e
136 log145.jpg    690   150      f
137 log146.jpg    695   151      g
138 log147.jpg    700   152      h
139 log148.jpg    705   153      i
140 log149.jpg    710   154      j
141 log150.jpg    715   155      k
142 log151.jpg    720   156      l
143 log152.jpg    725   157      m
144 log153.jpg    730   158      n
145 log154.jpg    735   159      o
146 log155.jpg    740   160      p
147 log156.jpg    745   161      q
148 log157.jpg    750   162      r
149 log158.jpg    755   163      s
150 log159.jpg    760   164      t
151 log160.jpg    765   165      u
152 log161.jpg    770   166      v
153 log162.jpg    775   167      w
154 log163.jpg    780   168      x
155 log164.jpg    785   169      y
156 log165.jpg    790   170      z
157 log166.jpg    795   171      A
158 log167.jpg    800   172      B
159 log168.jpg    805   173      C
160 log169.jpg    810   174      D
161 log170.jpg    815   175      E
162 log171.jpg    820   176      F
163 log172.jpg    825   177      G
164 log173.jpg    830   178      H
165 log174.jpg    835   179      I
166 log175.jpg    840   180      J
167 log176.jpg    845   181      K
168 log177.jpg    850   182      L
169 log178.jpg    855   183      M
170 log179.jpg    860   184      N
171 log180.jpg    865   185      O
172 log181.jpg    870   186      P
173 log182.jpg    875   187      Q
174 log183.jpg    880   188      R
175 log184.jpg    885   189      S
176 log185.jpg    890   190      T
177 log186.jpg    895   191      U
178 log187.jpg    900   192      V
179 log188.jpg    905   193      W
180 log189.jpg    910   194      X
181 log190.jpg    915   195      Y
182 log191.jpg    920   196      Z
183 log192.jpg    925   197      a
184 log193.jpg    930   198      b
185 log194.jpg    935   199      c
186 log195.jpg    940   200      d
187 log196.jpg    945   201      e
188 log197.jpg    950   202      f
189 log198.jpg    955   203      g
190 log199.jpg    960   204      h
191 log200.jpg    965   205      i
192 log201.jpg    970   206      j
193 log202.jpg    975   207      k
194 log203.jpg    980   208      l
195 log204.jpg    985   209      m
196 log205.jpg    990   210      n
197 log206.jpg    995   211      o
198 log207.jpg   1000   212      p
199 log208.jpg   1005   213      q
200 log209.jpg   1010   214      r
201 log210.jpg   1015   215      s
202 log211.jpg   1020   216      t
203 log212.jpg   1025   217      u
204 log213.jpg   1030   218      v
205 log214.jpg   1035   219      w
206 log215.jpg   1040   220      x
207 log216.jpg   1045   221      y
208 log217.jpg   1050   222      z
209 log218.jpg   1055   223      A
210 log219.jpg   1060   224      B
211 log220.jpg   1065   225      C
212 log221.jpg   1070   226      D
213 log222.jpg   1075   227      E
214 log223.jpg   1080   228      F
215 log224.jpg   1085   229      G
216 log225.jpg   1090   230      H
217 log226.jpg   1095   231      I
218 log227.jpg   1100   232      J
219 log228.jpg   1105   233      K
220 log229.jpg   1110   234      L
221 log230.jpg   1115   235      M
222 log231.jpg   1120   236      N
223 log232.jpg   1125   237      O
224 log233.jpg   1130   238      P
225 log234.jpg   1135   239      Q
226 log235.jpg   1140   240      R
227 log236.jpg   1145   241      S
228 log237.jpg   1150   242      T
229 log238.jpg   1155   243      U
230 log239.jpg   1160   244      V
231 log240.jpg   1165   245      W
232 log241.jpg   1170   246      X
233 log242.jpg   1175   247      Y
234 log243.jpg   1180   248      Z
235 log244.jpg   1185   249      a
236 log245.jpg   1190   250      b
237 log246.jpg   1195   251      c
238 log247.jpg   1200   252      d
239 log248.jpg   1205   253      e
240 log249.jpg   1210   254      f
241 log250.jpg   1215   255      g
242 log251.jpg   1220   256      h
243 log252.jpg   1225   257      i
244 log253.jpg   1230   258      j
245 log254.jpg   1235   259      k
246 log255.jpg   1240   260      l
247 log256.jpg   1245   261      m
248 log257.jpg   1250   262      n
249 log258.jpg   1255   263      o
250 log259.jpg   1260   264      p
251 log260.jpg   1265   265      q
252 log261.jpg   1270   266      r
253 log262.jpg   1275   267      s
254 log263.jpg   1280   268      t
255 log264.jpg   1285   269      u
256 log265.jpg   1290   270      v
257 log266.jpg   1295   271      w
258 log267.jpg   1300   272      x
259 log268.jpg   1305   273      y
260 log269.jpg   1310   274      z
261 log270.jpg   1315   275      A
262 log271.jpg   1320   276      B
263 log272.jpg   1325   277      C
264 log273.jpg   1330   278      D
265 log274.jpg   1335   279      E
266 log275.jpg   1340   280      F
267 log276.jpg   1345   281      G
268 log277.jpg   1350   282      H
269 log278.jpg   1355   283      I
270 log279.jpg   1360   284      J
271 log280.jpg   1365   285      K
272 log281.jpg   1370   286      L
273 log282.jpg   1375   287      M
274 log283.jpg   1380   288      N
275 log284.jpg   1385   289      O
276 log285.jpg   1390   290      P
277 log286.jpg   1395   291      Q
278 log287.jpg   1400   292      R
279 log288.jpg   1405   293      S
280 log289.jpg   1410   294      T
281 log290.jpg   1415   295      U
282 log291.jpg   1420   296      V
283 log292.jpg   1425   297      W
284 log293.jpg   1430   298      X
285 log294.jpg   1435   299      Y
286 log295.jpg   1440   300      Z
287 log296.jpg   1445   301      a
288 log297.jpg   1450   302      b
289 log298.jpg   1455   303      c
290 log299.jpg   1460   304      d
291 log300.jpg   1465   305      e
292 log301.jpg   1470   306      f
293 log302.jpg   1475   307      g
294 log303.jpg   1480   308      h
295 log304.jpg   1485   309      i
296 log305.jpg   1490   310      j
297 log306.jpg   1495   311      k
298 log307.jpg   1500   312      l
299 log308.jpg   1505   313      m
300 log309.jpg   1510   314      n
301 log310.jpg   1515   315      o
302 log311.jpg   1520   316      p
303 log312.jpg   1525   317      q
304 log313.jpg   1530   318      r
305 log314.jpg   1535   319      s
306 log315.jpg   1540   320      t
307 log316.jpg   1545   321      u
308 log317.jpg   1550   322      v
309 log318.jpg   1555   323      w
310 log319.jpg   1560   324      x
311 log320.jpg   1565   325      y
312 log321.jpg   1570   326      z
313 log322.jpg   1575   327      A
314 log323.jpg   1580   328      B
315 log324.jpg   1585   329      C
316 log325.jpg   1590   330      D
317 log326.jpg   1595   331      E
318 log327.jpg   1600   332      F
319 log328.jpg   1605   333      G
320 log329.jpg   1610   334      H
321 log330.jpg   1615   335      I
322 log331.jpg   1620   336      J
323 log332.jpg   1625   337      K
324 log333.jpg   1630   338      L
325 log334.jpg   1635   339      M
326 log335.jpg   1640   340      N
327 log336.jpg   1645   341      O
328 log337.jpg   1650   342      P
329 log338.jpg   1655   343      Q
330 log339.jpg   1660   344      R
331 log340.jpg   1665   345      S
332 log341.jpg   1670   346      T
333 log342.jpg   1675   347      U
334 log343.jpg   1680   348      V
335 log344.jpg   1685   349      W
336 log345.jpg   1690   350      X
337 log346.jpg   1695   351      Y
338 log347.jpg   1700   352      Z
339 log348.jpg   1705   353      a
340 log349.jpg   1710   354      b
341 log350.jpg   1715   355      c
342 log351.jpg   1720   356      d
343 log352.jpg   1725   357      e
344 log353.jpg   1730   358      f
345 log354.jpg   1735   359      g
346 log355.jpg   1740   360      h
347 log356.jpg   1745   361      i
348 log357.jpg   1750   362      j
349 log358.jpg   1755   363      k
350 log359.jpg   1760   364      l
351 log360.jpg   1765   365      m
352 log361.jpg   1770   366      n
353 log362.jpg   1775   367      o
354 log363.jpg   1780   368      p
355 log364.jpg   1785   369      q
356 log365.jpg   1790   370      r
357 log366.jpg   1795   371      s
358 log367.jpg   1800   372      t
359 log368.jpg   1805   373      u
360 log369.jpg   1810   374      v
361 log370.jpg   1815   375      w
362 log371.jpg   1820   376      x
363 log372.jpg   1825   377      y
364 log373.jpg   1830   378      z
365 log374.jpg   1835   379      A
366 log375.jpg   1840   380      B
367 log376.jpg   1845   381      C
368 log377.jpg   1850   382      D
369 log378.jpg   1855   383      E
370 log379.jpg   1860   384      F
371 log380.jpg   1865   385      G
372 log381.jpg   1870   386      H
373 log382.jpg   1875   387      I
374 log383.jpg   1880   388      J
375 log384.jpg   1885   389      K
376 log385.jpg   1890   390      L
377 log386.jpg   1895   391      M
378 log387.jpg   1900   392      N
379 log388.jpg   1905   393      O
380 log389.jpg   1910   394      P
381 log390.jpg   1915   395      Q
382 log391.jpg   1920   396      R
383 log392.jpg   1925   397      S
384 log393.jpg   1930   398      T
385 log394.jpg   1935   399      U
386 log395.jpg   1940   400      V
387 log396.jpg   1945   401      W
388 log397.jpg   1950   402      X
389 log398.jpg   1955   403      Y
390 log399.jpg   1960   404      Z
391 log400.jpg   1965   405      a
392 log401.jpg   1970   406      b
393 log402.jpg   1975   407      c
394 log403.jpg   1980   408      d
395 log404.jpg   1985   409      e
396 log405.jpg   1990   410      f
397 log406.jpg   1995   411      g
398 log407.jpg   2000   412      h
399 log408.jpg   2005   413      i
400 log409.jpg   2010   414      j
401 log410.jpg   2015   415      k
402 log411.jpg   2020   416      l
403 log412.jpg   2025   417      m
404 log413.jpg   2030   418      n
405 log414.jpg   2035   419      o
406 log415.jpg   2040   420      p
407 log416.jpg   2045   421      q
408 log417.jpg   2050   422      r
409 log418.jpg   2055   423      s
410 log419.jpg   2060   424      t
411 log420.jpg   2065   425      u
412 log421.jpg   2070   426      v
413 log422.jpg   2075   427      w
414 log423.jpg   2080   428      x
415 log424.jpg   2085   429      y
416 log425.jpg   2090   430      z
417 log426.jpg   2095   431      A
418 log427.jpg   2100   432      B
419 log428.jpg   2105   433      C
420 log429.jpg   2110   434      D
421 log430.jpg   2115   435      E
422 log431.jpg   2120   436      F
423 log432.jpg   2125   437      G
424 log433.jpg   2130   438      H
425 log434.jpg   2135   439      I
426 log435.jpg   2140   440      J
427 log436.jpg   2145   441      K
428 log437.jpg   2150   442      L
429 log438.jpg   2155   443      M
430 log439.jpg   2160   444      N
431 log440.jpg   2165   445      O
432 log441.jpg   2170   446      P
433 log442.jpg   2175   447      Q
434 log443.jpg   2180   448      R
435 log444.jpg   2185   449      S
436 log445.jpg   2190   450      T
437 log446.jpg   2195   451      U
438 log447.jpg   2200   452      V
439 log448.jpg   2205   453      W
440 log449.jpg   2210   454      X
441 log450.jpg   2215   455      Y
442 log451.jpg   2220   456      Z
443 log452.jpg   2225   457      a
444 log453.jpg   2230   458      b
445 log454.jpg   2235   459      c
446 log455.jpg   2240   460      d
447 log456.jpg   2245   461      e
448 log457.jpg   2250   462      f
449 log458.jpg   2255   463      g
450 log459.jpg   2260   464      h
451 log460.jpg   2265   465      i
452 log461.jpg   2270   466      j
453 log462.jpg   2275   467      k
454 log463.jpg   2280   468      l
455 log464.jpg   2285   469      m
456 log465.jpg   2290   470      n
457 log466.jpg   2295   471      o
458 log467.jpg   2300   472      p
459 log468.jpg   2305   473      q
460 log469.jpg   2310   474      r
461 log470.jpg   2315   475      s
462 log471.jpg   2320   476      t
463 log472.jpg   2325   477      u
464 log473.jpg   2330   478      v
465 log474.jpg   2335   479      w
466 log475.jpg   2340   480      x
467 log476.jpg   2345   481      y
468 log477.jpg   2350   482      z
469 log478.jpg   2355   483      A
470 log479.jpg   2360   484      B
471 log480.jpg   2365   485      C
472 log481.jpg   2370   486      D
473 log482.jpg   2375   487      E
474 log483.jpg   2380   488      F
475 log484.jpg   2385   489      G
476 log485.jpg   2390   490      H
477 log486.jpg   2395   491      I
478 log487.jpg   2400   492      J
479 log488.jpg   2405   493      K
480 log489.jpg   2410   494      L
481 log490.jpg   2415   495      M
482 log491.jpg   2420   496      N
483 log492.jpg   2425   497      O
484 log493.jpg   2430   498      P
485 log494.jpg   2435   499      Q
486 log495.jpg   2440   500      R
487 log496.jpg   2445   501      S
488 log497.jpg   2450   502      T
489 log498.jpg   2455   503      U
490 log499.jpg   2460   504      V
491 log500.jpg   2465   505      W
492 log501.jpg   2470   506      X
493 log502.jpg   2475   507      Y
494 log503.jpg   2480   508      Z
495 log504.jpg   2485   509      a
496 log505.jpg   2490   510      b
497 log506.jpg   2495   511      c
498 log507.jpg   2500   512      d
499 log508.jpg   2505   513      e
500 log509.jpg   2510   514      f
501 log510.jpg   2515   515      g
502 log511.jpg   2520   516      h
503 log512.jpg   2525   517      i
504 log513.jpg   2530   518      j
505 log514.jpg   2535   519      k
506 log515.jpg   2540   520      l
507 log516.jpg   2545   521      m
508 log517.jpg   2550   522      n
509 log518.jpg   2555   523      o
510 log519.jpg   2560   524      p
511 log520.jpg   2565   525      q
512 log521.jpg   2570   526      r
513 log522.jpg   2575   527      s
514 log523.jpg   2580   528      t
515 log524.jpg   2585   529      u
516 log525.jpg   2590   530      v
517 log526.jpg   2595   531      w
518 log527.jpg   2600   532      x
519 log528.jpg   2605   533      y
520 log529.jpg   2610   534      z
521 log530.jpg   2615   535      A
522 log531.jpg   2620   536      B
523 log532.jpg   2625   537      C
524 log533.jpg   2630   538      D
525 log534.jpg   2635   539      E
526 log535.jpg   2640   540      F
527 log536.jpg   2645   541      G
528 log537.jpg   2650   542      H
529 log538.jpg   2655   543      I
530 log539.jpg   2660   544      J
531 log540.jpg   2665   545      K
532 log541.jpg   2670   546      L
533 log542.jpg   2675   547      M
534 log543.jpg   2680   548      N
535 log544.jpg   2685   549      O
536 log545.jpg   2690   550      P
537 log546.jpg   2695   551      Q
538 log547.jpg   2700   552      R
539 log548.jpg   2705   553      S
540 log549.jpg   2710   554      T
541 log550.jpg   2715   555      U
542 log551.jpg   2720   556      V
543 log552.jpg   2725   557      W
544 log553.jpg   2730   558      X
545 log554.jpg   2735   559      Y
546 log555.jpg   2740   560      Z
547 log556.jpg   2745   561      a
548 log557.jpg   2750   562      b
549 log558.jpg   2755   563      c
550 log559.jpg   2760   564      d
551 log560.jpg   2765   565      e
552 log561.jpg   2770   566      f
553 log562.jpg   2775   567      g
554 log563.jpg   2780   568      h
555 log564.jpg   2785   569      i
556 log565.jpg   2790   570      j
557 log566.jpg   2795   571      k
558 log567.jpg   2800   572      l
559 log568.jpg   2805   573      m
560 log569.jpg   2810   574      n
561 log570.jpg   2815   575      o
562 log571.jpg   2820   576      p
563 log572.jpg   2825   577      q
564 log573.jpg   2830   578      r
565 log574.jpg   2835   579      s
566
```

Simple Simulation:



Full Simulation:



3. Conclusion

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