UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Department of Electronic & Computer Engineering

Name and ID:	Bartholomew O'Keeffe Mark O'Neill Lina Albay	14180847 14117711 14118432
Course:	LM 083 BSc in Mobile Communications and Security	
Module Code:	ET4027	
Module Title:	Computer Forensics	
Assignment:	Module Assignment (Group 7)	
Date:	March 05, 2018	
Lecturer:	Dr. Jacqueline Walker	

Objectives:

- Learn how to uses a disk hex editor to explore a disk drive
- Learn how ot use an imaging tool to image disk drives and mount images
- Learn one disk partition scheme in detail
- Learn the FAT file system in detail
- Learn the NTFS file system in detail
- Learn how to write a basic file system forensic tool

Assumptions:

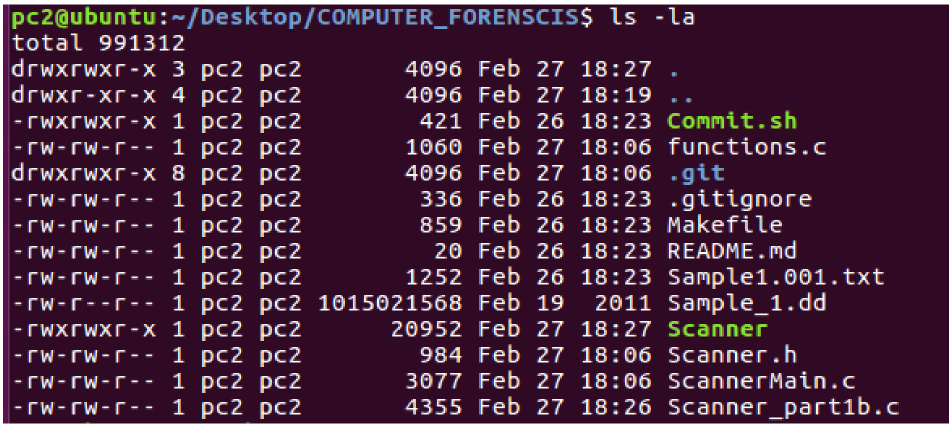
1) Use of **Sample_1.dd** image file downloaded from \\dell24\ee6012et4027 folder from internal UL machine.
2) Assumes that the forensic tool can only be used on a disk drive image that has a standard MBR with the first partition formatted as FA-16 only.

Requirements and Description of solutions:

N.B: This section of the project report contains screen capture extracted in running the final script for this project. All program code used were attached at the end of this report.

Figure 1 contains the files created in accomplishing our forensic tool. Below are the files needed in prior to execution:

a) **Sample_1.dd** – disk image used for investigation
b) **Scanner.h** – contains header file that separate certain elements of a program's source code int reusable files.
c) **ScannerMain.c** – source code that contains the program start up, functions and methods design to read and write information for partition, FAT volume and NTFS volume info.
d) **Scanner_part1b.c** – code for analysing FAT-16 volume and recovering the deleted file.
e) **Makefile** – contains specification on how to derive the target program that runs **make** utility to automatically builds executable programs and libraries.



```
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$ ls -la
total 991312
drwxrwxr-x 3 pc2 pc2       4096 Feb 27 18:27 .
drwxr-xr-x 4 pc2 pc2       4096 Feb 27 18:19 ..
-rwxrwxr-x 1 pc2 pc2        421 Feb 26 18:23 Commit.sh
-rw-rw-r-- 1 pc2 pc2       1060 Feb 27 18:06 functions.c
drwxrwxr-x 8 pc2 pc2       4096 Feb 27 18:06 .git
-rw-rw-r-- 1 pc2 pc2        336 Feb 26 18:23 .gitignore
-rw-rw-r-- 1 pc2 pc2        859 Feb 26 18:23 Makefile
-rw-rw-r-- 1 pc2 pc2         20 Feb 26 18:23 README.md
-rw-rw-r-- 1 pc2 pc2       1252 Feb 26 18:23 Sample1.001.txt
-rw-r--r-- 1 pc2 pc2 1015021568 Feb 19  2011 Sample_1.dd
-rwxrwxr-x 1 pc2 pc2      20952 Feb 27 18:27 Scanner
-rw-rw-r-- 1 pc2 pc2        984 Feb 27 18:06 Scanner.h
-rw-rw-r-- 1 pc2 pc2       3077 Feb 27 18:06 ScannerMain.c
-rw-rw-r-- 1 pc2 pc2       4355 Feb 27 18:26 Scanner_part1b.c
```

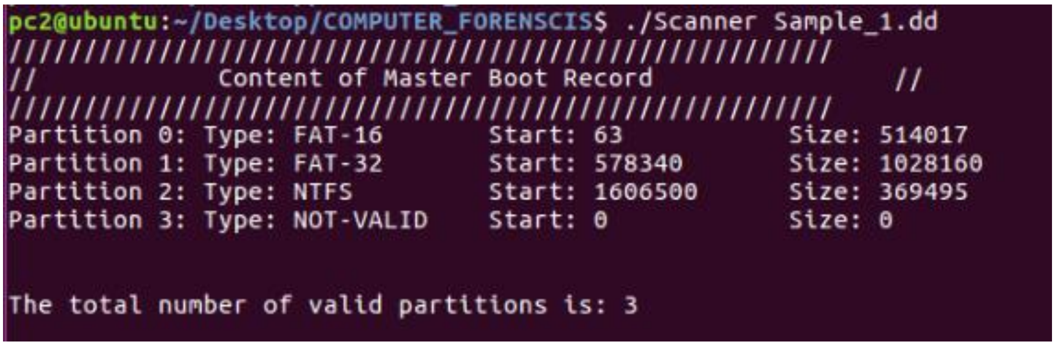
Figure 1: Files created that comprises the forensic tool project

Write a program to do the following:

Required:

a) **Partition information** – display the number of partitions on the disk and or each partition display the start sector, size of partition and file system type.

Solution: Partition information output capture:



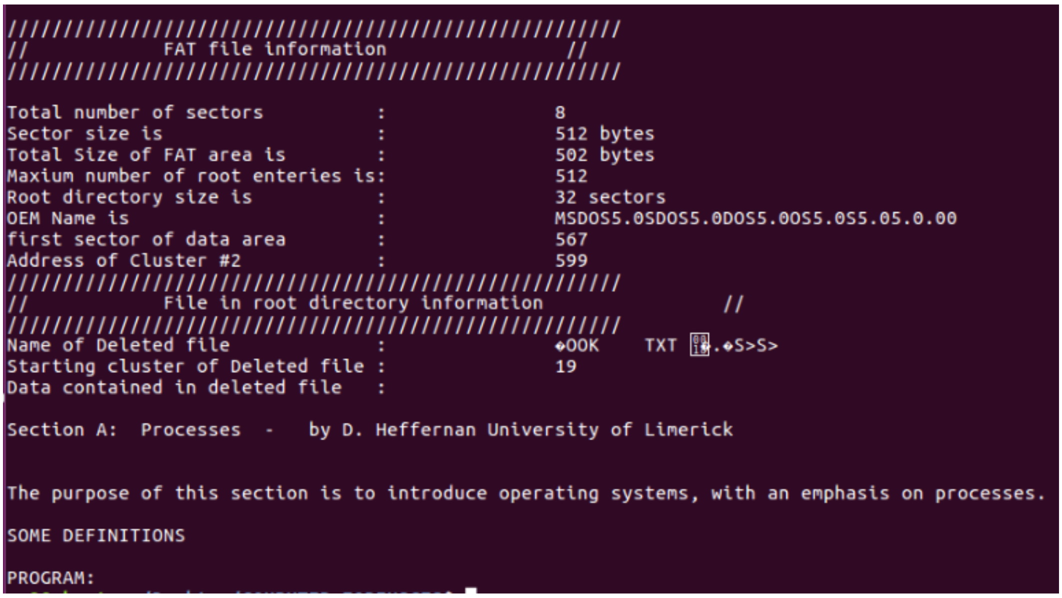
```
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$ ./Scanner Sample_1.dd
//////////////////////////////////////////////////////
//             Content of Master Boot Record              //
//////////////////////////////////////////////////////
Partition 0: Type: FAT-16       Start: 63           Size: 514017
Partition 1: Type: FAT-32       Start: 578340       Size: 1028160
Partition 2: Type: NTFS         Start: 1606500      Size: 369495
Partition 3: Type: NOT-VALID    Start: 0            Size: 0


The total number of valid partitions is: 3
```

Required:

b) **Fat Volume information** – for this partition only, display the number of sectors per cluster, the size of the FAT area, the size of the Root Directory, and the sector address of Cluster #2.

For the first deleted file on the volume's root directory, display the name and size of that file, and the number of the first cluster. Display the first 16 characters of the content of that file (assume it as a simple text file).

Solution: Fat Volume information and First deleted file on Root directory output capture:

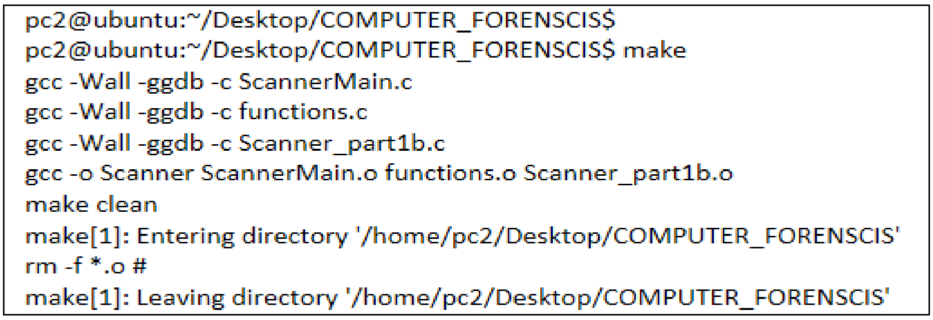


```
////////////////////////////////////////////////////////
//            FAT file information                  //
////////////////////////////////////////////////////////

Total number of sectors          :               8
Sector size is                   :               512 bytes
Total Size of FAT area is        :               502 bytes
Maxium number of root enteries is:               512
Root directory size is           :               32 sectors
OEM Name is                      :               MSDOS5.0SDOS5.0DOS5.0OS5.0S5.05.0.00
first sector of data area        :               567
Address of Cluster #2            :               599
////////////////////////////////////////////////////////
//            File in root directory information                //
////////////////////////////////////////////////////////
Name of Deleted file             :               �OOK    TXT ▯�.�S>S>
Starting cluster of Deleted file :               19
Data contained in deleted file   :

Section A:  Processes  -   by D. Heffernan University of Limerick


The purpose of this section is to introduce operating systems, with an emphasis on processes.

SOME DEFINITIONS

PROGRAM:
```

Description of solutions:

- Source code for Group 7 is available on request on the following links:
  - https://github.com/barthskywalker/COMPUTER_FORENSCIS
  - https://github.com/linaalbay/COMPUTER_FORENSCIS
- Clone the COMPUTER_FORENSCIS folder to obtain all the program needed.
- Save this folder to your Linux machine.
- Make sure that the disc image, Sample_1.dd is in the same folder (disc image to analyse).
- Login to your Linux machine and navigate to COMPUTER_FORENSCIS folder.
- Once done, execute the "**make**" command (just type **make** on your terminal)

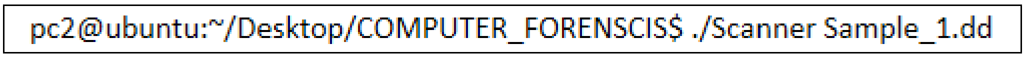
  You should observe similar result below:



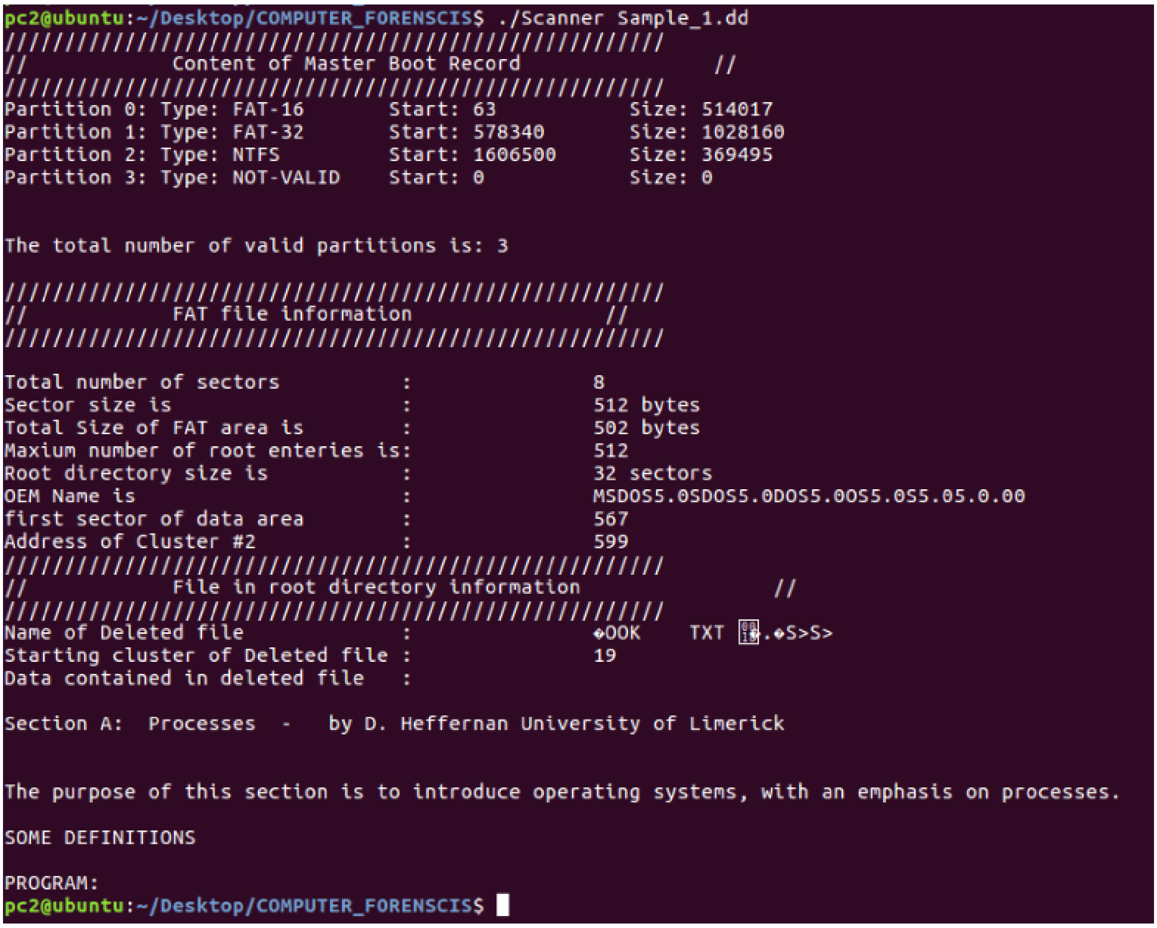
```
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$ make
gcc -Wall -ggdb -c ScannerMain.c
gcc -Wall -ggdb -c functions.c
gcc -Wall -ggdb -c Scanner_part1b.c
gcc -o Scanner ScannerMain.o functions.o Scanner_part1b.o
make clean
make[1]: Entering directory '/home/pc2/Desktop/COMPUTER_FORENSCIS'
rm -f *.o #
make[1]: Leaving directory '/home/pc2/Desktop/COMPUTER_FORENSCIS'
```

- Then run the executable file: Use the command: **./Scanner Sample_1.dd**

  You should observe similar result below where FAT file and File in root directory information is printed:



```
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$ ./Scanner Sample_1.dd
```



```
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$ ./Scanner Sample_1.dd
////////////////////////////////////////////////////////
//            Content of Master Boot Record                //
////////////////////////////////////////////////////////
Partition 0: Type: FAT-16       Start: 63           Size: 514017
Partition 1: Type: FAT-32       Start: 578340       Size: 1028160
Partition 2: Type: NTFS         Start: 1606500      Size: 369495
Partition 3: Type: NOT-VALID    Start: 0            Size: 0


The total number of valid partitions is: 3

////////////////////////////////////////////////////////
//            FAT file information                //
////////////////////////////////////////////////////////

Total number of sectors          :                8
Sector size is                   :                512 bytes
Total Size of FAT area is        :                502 bytes
Maxium number of root enteries is:                512
Root directory size is           :                32 sectors
OEM Name is                      :                MSDOS5.0SDOS5.0DOS5.0OS5.0S5.05.0.00
first sector of data area        :                567
Address of Cluster #2            :                599
////////////////////////////////////////////////////////
//            File in root directory information                //
////////////////////////////////////////////////////////
Name of Deleted file             :                �OOK    TXT ▯�.�S>S>
Starting cluster of Deleted file :                19
Data contained in deleted file   :

Section A:  Processes  -   by D. Heffernan University of Limerick


The purpose of this section is to introduce operating systems, with an emphasis on processes.

SOME DEFINITIONS

PROGRAM:
pc2@ubuntu:~/Desktop/COMPUTER_FORENSCIS$ █
```

Statement of completion:

This report is an accomplishement of Group 7 Phase 1 and Phase 2 – Full tool. Source code are attached and instructions in accessing and running the executable file created was outlined together with screen captures.

Source Code:

HEADER FILE

```
/**
 * Header file for Computer Forensics Project
 * StudentName&ID: Barth O'Keeffe    14180847
 *                 Mark O’Neill      14117711
 *                 Lina Albay        14118432
 * Lecturer:       Dr. Jacqueline Walker

 * Date  16/02/2018
 */

#ifndef Scanner_H
#define Scanner_H

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

/*
   globals
 */
//File pionter
FILE *fp;

/**
 * structure declarations
 */
struct Partition
{
        char type;
        int start_sect;
        int size;
} part_entry[4]; // 4 x partition table entry
/**
 * struct fat FILEsystem
 */
struct FAT
{
        int Num_sectors;
        int sector_size;
        int FAT_size;
        int Max_root_enteries;
        int rootSize;
        char *OEM_Name;
        int Sector_address_Data_Area;
        int address_of_Cluster_2;

} fat;
/**
 * structure to store deleted file info
 */
struct Deleted_file
{
        char* name;
        char* contents;
        int file_size;
        int Starting_cluster;
}DelFile;
/**
 * Function declarations;
 */
void getFatinfo();
char *concat(const char *s1, const char *s2);
void print_Header(char *s);
char *getOEM(const char *s1);
void getFileFromFatInfo();
#endif
```

MAIN FILE

```c
/**
 * ScannerMain.c file for Computer Forensics Project
 * StudentName&ID: Barth O'Keeffe    14180847
 *                 Mark O’Neill      14117711
 *                 Lina Albay        14118432
 * Lecturer:       Dr. Jacqueline Walker
 * Date  16/02/2018
 */
///////////////////////////////////////////////////////////
//ScannerMain.c
//////////////////////////////////////////////////////////

#include "Scanner.h"
int main(int argc, char *argv[])
{
        // Define some variables
        int i, offset = 16, not_exist = 0;
        char buf_part_table[64], vol_type[12];
        fp = fopen(argv[1], "rb");         // Open file for reading - binary mode. Should
use error check!
        fseek(fp, 0x1BE, SEEK_SET);        // Seek to the start of the part_entry list
        fread(buf_part_table, 1, 64, fp); // Read the 64-byte block to memory buffer
        print_Header("Content of Master Boot Record");
        for (i = 0; i < 4; i++)
        {
                part_entry[i].type = *(char *)(buf_part_table + 0x04 + (i * offset));
                if (part_entry[i].type == 0)
                        not_exist++;
                part_entry[i].start_sect = *(int *)(buf_part_table + 0x08 + (i *
offset));
                part_entry[i].size = *(int *)(buf_part_table + 0x0C + (i * offset));
                switch (part_entry[i].type)
                {
                case 00:
                        strcpy(vol_type, "NOT-VALID");
                        break;
                case 06:
                        strcpy(vol_type, "FAT-16");
                        break;
                case 07:
                        strcpy(vol_type, "NTFS");
                        break;
                case 0x0B:
                        strcpy(vol_type, "FAT-32");
                        break;
                default:
                        strcpy(vol_type, "NOT-DECODED");
                        break;
                }

                // Print out partition content
                printf("Partition %d: Type: %-12s Start: %-12d Size: %-12d\n", i,
vol_type, part_entry[i].start_sect,
                       part_entry[i].size);
        }
        getFatinfo();
        getFileFromFatInfo();
        printf("\n\nThe total number of valid partitions is: %d\n\n", (4 - not_exist));
        print_Header("FAT file information");
        printf("\n");
        printf("Total number of sectors          :\t\t %d\n", fat.Num_sectors);
        printf("Sector size is                   :\t\t %d bytes\n", fat.sector_size);
        printf("Total Size of FAT area is        :\t\t %d bytes\n", fat.FAT_size);
        printf("Maxium number of root enteries is:\t\t %d\n", fat.Max_root_enteries);
        printf("Root directory size is           :\t\t %d sectors\n", fat.rootSize);
        printf("OEM Name is                      :\t\t %s\n", fat.OEM_Name);
        printf("first sector of data area        :\t\t %d\n", 
fat.Sector_address_Data_Area);
        printf("Address of Cluster #2            :\t\t %d\n", fat.address_of_Cluster_2);
        print_Header("File in root directory information");
        printf("Name of Deleted file             :\t\t %s\n", DelFile.name);
        printf("Starting cluster of Deleted file :\t\t %d\n", DelFile.Starting_cluster);
        printf("Data contained in deleted file   :\t\t%s\n", DelFile.contents);
        fclose(fp);
        return (0);
}
```

SCANNER_PART1B FILE

```
#include "Scanner.h"
/**
 * Pass the structure array that is the MBR and returns the details of FAT Partition
 * as the structure FAT
 * @param  a place holder for the partition type array
 * @return   The FAT info is returned as a structure
 */
void getFatinfo()
{
        char buf_part_table[100];
        int startsect = part_entry[0].start_sect * 512;
        fseek(fp, startsect, SEEK_SET); // Seek to the start of the part_entry list
        fread(buf_part_table, 1, 100, fp);
        fat.Num_sectors = *(buf_part_table + 0x0D);
        /*code to get the total size of fat Area*/
        int Num_copys_of_FAT = *(buf_part_table + 0x10);
        char temp_buff[2];
        fseek(fp, startsect + 0x16, SEEK_SET); // Seek to the start of the part_entry
list
        fread(temp_buff, 1, 2, fp);
        short size_of_FAT_in_Sectors1 = *(int *)(temp_buff);
        int size_of_FAT_in_Sectors = size_of_FAT_in_Sectors1;
        /*FAT Area size = (size of FAT in sectors) * (number of FAT copies)*/
        fat.FAT_size = Num_copys_of_FAT * size_of_FAT_in_Sectors;
        /*code to get the total size of Root Directory*/
        char temp_buff1[2];
        fseek(fp, startsect + 0x11, SEEK_SET); // Seek to the start of the part_entry
list
        fread(temp_buff1, 1, 2, fp);
        int Max_num_root_enteries1 = *(int *)(temp_buff1);
        //returns the max number of root directory entries
        fat.Max_root_enteries = Max_num_root_enteries1;
        //get size of a sector in bytes
        char temp_buff2[2];
        fseek(fp, startsect + 0x0B, SEEK_SET); // Seek to the start of the part_entry
list
        fread(temp_buff2, 1, 2, fp);
        int size_of_a_sector1 = *(int *)(temp_buff2);
        fat.sector_size = size_of_a_sector1;

        /*Root dir size = ( max no. of dir entries) * (dir entry size in bytes) / sector
size*/
        fat.rootSize = fat.Max_root_enteries * 32 / fat.sector_size;
        fat.OEM_Name = getOEM(buf_part_table);
        /*
                 Sector address (Data Area)
                 = (first sector of volume) + (size of reserved area) + (size of FAT
Area)
         */
        //get size of reserved Area
        int reserved_Area = *(buf_part_table + 0x0E);
        fat.Sector_address_Data_Area = part_entry[0].start_sect + reserved_Area +
fat.FAT_size;
        /*
                 Cluster #2 address = ( first sector of data area ) + ( size of root
directory)
         */
        fat.address_of_Cluster_2 = fat.Sector_address_Data_Area + fat.rootSize;
}



void getFileFromFatInfo()
{
        int start = fat.Sector_address_Data_Area*fat.sector_size; // get starting point
of root directory
        int buff_size= part_entry[0].size; //get size of root directory in bytes
        char root_buff[buff_size]; //buffer to hold root directory contents
        fseek(fp, start, SEEK_SET); // Seek to the start of the root directory
        fread(root_buff, 1, buff_size, fp); //read contents of root directory into buffer
        int location_of_deleted_file_name;
        //find deleted file 0xE5
        for (int i = 0; i < buff_size; i++) {
              //  printf("%x\n",*root_buff+i);
                if(*root_buff+i==0xE5){
                location_of_deleted_file_name=i;
              }
        }
        //root_buff+location_of_deleted_file_name-(11*8) to return to the start of the
filename
        int start_point_for_name=location_of_deleted_file_name-(11*8);
        DelFile.name=root_buff+start_point_for_name;
        /*
        get starting cluster
         */
        int Starting_cluster1 = *(root_buff+start_point_for_name+0x1A);
        DelFile.Starting_cluster=Starting_cluster1;
        /*
        CSA = (sector address for Cluster #2) + ( (cluster number – 2) * 8 )
         */
        int CSA =fat.address_of_Cluster_2 +((DelFile.Starting_cluster-2)*8);
        int start_of_deleted_data=fat.sector_size*CSA;
        //create temperory buffer to store first 16 characters
        char temp_buff2[200];
        fseek(fp, start_of_deleted_data, SEEK_SET); // Seek to first 16 characters
        fread(temp_buff2, 1, 200, fp);//read in first 16 characters
        DelFile.contents=malloc(strlen(temp_buff2)+1);//make sure structure can hold data
        DelFile.contents=strcpy(DelFile.contents,temp_buff2);//copy data to structure

      }
```

FUNCTIONS FILE

```c
#include "Scanner.h"

/**
 * Function to concatanate two string values
 * @param  s1 first string
 * @param  s2
 * @return    return concatanated string
 */
char *concat(const char *s1, const char *s2)
{
  char *result = malloc(strlen(s1) + strlen(s2) + 1); //+1 for the null-terminator
  strcpy(result, s1);
  strcat(result, s2);
  return result;
}

/**
 * prints the OEM name to the screen
 * @param  s1 pointer to start of OEM
 * @return    returns full oem name
 */
char *getOEM(const char *s1)
{
  char *oem;
  oem = concat((s1 + 0x03), (s1 + 0x04));
  oem = concat(oem, (s1 + 0x05));
  oem = concat(oem, (s1 + 0x06));
  oem = concat(oem, (s1 + 0x07));
  oem = concat(oem, (s1 + 0x08));
  oem = concat(oem, (s1 + 0x09));
  oem = concat(oem, (s1 + 0x0A));
  return oem;
}

/**
 * print out headers
 * @param s [description]
 */
void print_Header(char *s)
{

  printf("///////////////////////////////////////////////////////\n");
  printf("//            %s                //\n", s);
  printf("///////////////////////////////////////////////////////\n");
}
```

MAKE FILE

```
#################################################
# Makefile for Computer Forensics Project      #
# StudentName&ID: Barth O'Keeffe    14180847   #
#               Mark O’Neill      14117711   #
#               Lina Albay       14118432   #
# Lecturer:       Dr. Jacqueline Walker       #
# Date  16/02/2018                          #
#################################################

HEADER  = Scanner.h

#command to link objects and create excutable
Scanner: ScannerMain.o functions.o Scanner_part1b.o
	gcc -o Scanner ScannerMain.o functions.o Scanner_part1b.o
	make clean

#compile additional source files into object file
##### Add Here !!!!
#compile functions.c files
functions.o: functions.c
	gcc -Wall -ggdb -c functions.c

#compile Scanner_part1b.c file
Scanner_part1b.o: Scanner_part1b.c
	gcc -Wall -ggdb -c Scanner_part1b.c

# compile main application file
ScannerMain.o: ScannerMain.c
	gcc -Wall -ggdb -c ScannerMain.c

# clean up - handy to enforce recompilationls
clean:
	rm -f *.o #
```

COMMIT FILE

```
#!/bin/bash
#01-06-2017 by: Assignment Group 7
 ##########################################################
 ##            Script to automate pushing to git                              #
 ##########################################################

 git add --all --verbose
 #get commit message form user
 echo "Please enter commit message \n"
 read -r Message
 # command to commit to git
 git commit -m "$Message"
 # finally push code
 git push
```