UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

# Department of Electronic & Computer Engineering



| | | |
|---|---|---|
| **Name and ID:** | Bartholomew O'Keeffe | 14180847 |
| | Mark O'Neill | 14117711 |
| | Lina Albay | 14118432 |



| | |
|---|---|
| **Course:** | LM 083 BSc in Mobile Communications and Security |
| **Module Code:** | ET4027 |
| **Module Title:** | Computer Forensics |
| **Assignment:** | Assignment1_Phase2 (Group 7) |
| **Date:** | March 31, 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.

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.

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).

c) **NTFS Volume information** – display information for an NTFS partition as follow:

- How many bytes per sector for this NTFS volume.
- How many sectors per cluster for this NTFS volume.
- What is the sector address for the $MFT file record.
- What is the type and length of the first two attributes in the $MFT record.

## 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 as highlighted in Figure1.
b) **Test.dd** – disk image created using FTK Imager Lite as highlight in Figure1.
c) **Scanner.h** – contains header file that separate certain elements of a program's source code int reusable files.
d) **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.
e) **Scanner_part1b.c** – source code for analysing FAT-16 volume and recovering the deleted file.
f) **Scanner_part1c.c** – source code for analysing NTFS partition.
g) **Makefile** – contains specification on how to derive the target program that runs **make** utility to automatically builds executable programs and libraries.

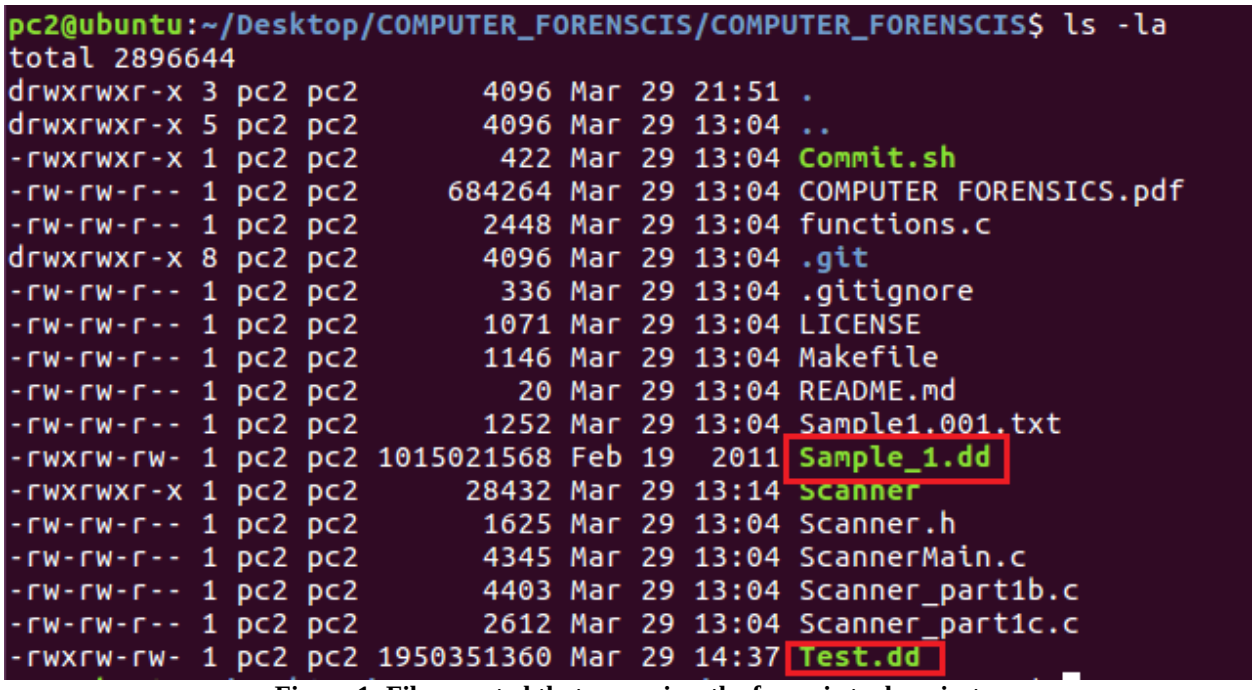


**Figure 1: Files created that comprises the forensic tool project**

## Steps to follow in running the Forensic Tool created:

- Source code for Group 7 is available through: https://github.com/barthskywalker/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 and Test.dd are both in the same folder (disc images to analyse).
- Login to your Linux machine and navigate to COMPUTER_FORENSCIS folder (where you saved the downloaded programs)
- Once done, execute the "**make**" command (just type **make** on your terminal)

You should observe similar result as shown in Figure2:

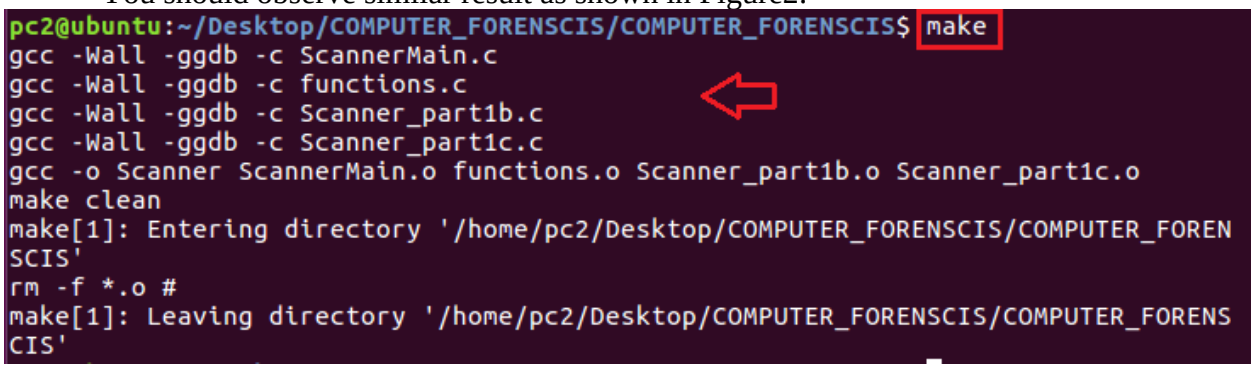


**Figure 2: make command output**

- Then run the executable file: Use the command: **./Scanner <insert the name of the dd file to analyse here>**

**N.B:** Part of this Assignment is to run the forensic tool created against a different disk image to verify its effectiveness. FTK Imager Lite was used to create a disk image using the filename **Test.dd** (results are captured as shown in Figure3)

## Test Result Screen Capture for a disk image created through FTK Imager:

## Command: ./Scanner Test

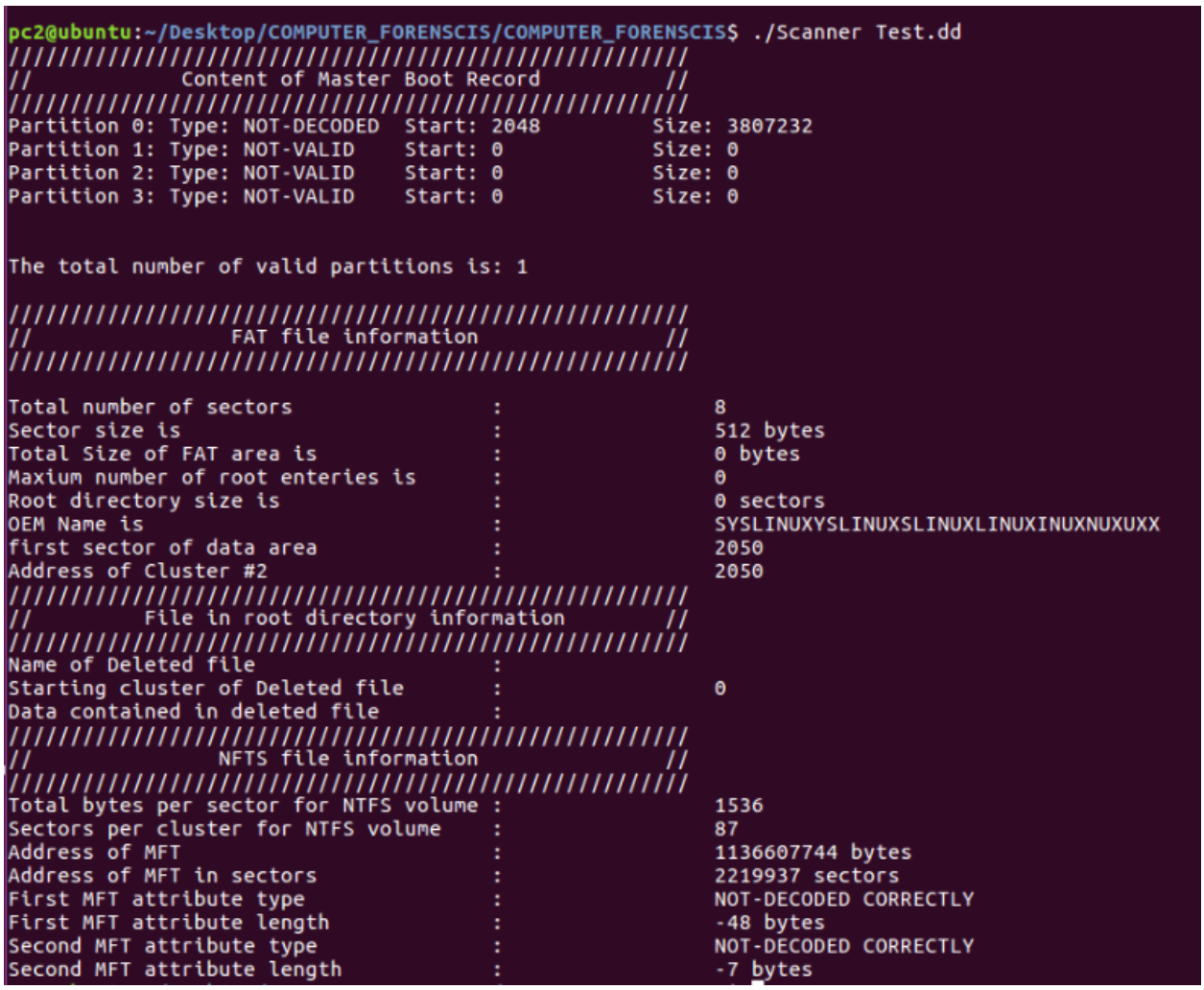


**Figure 3: disk image "Test.dd" output using the forensic tool created**

## Test Result Screen Capture for the full forensic tool created using disk image Sample_1.dd

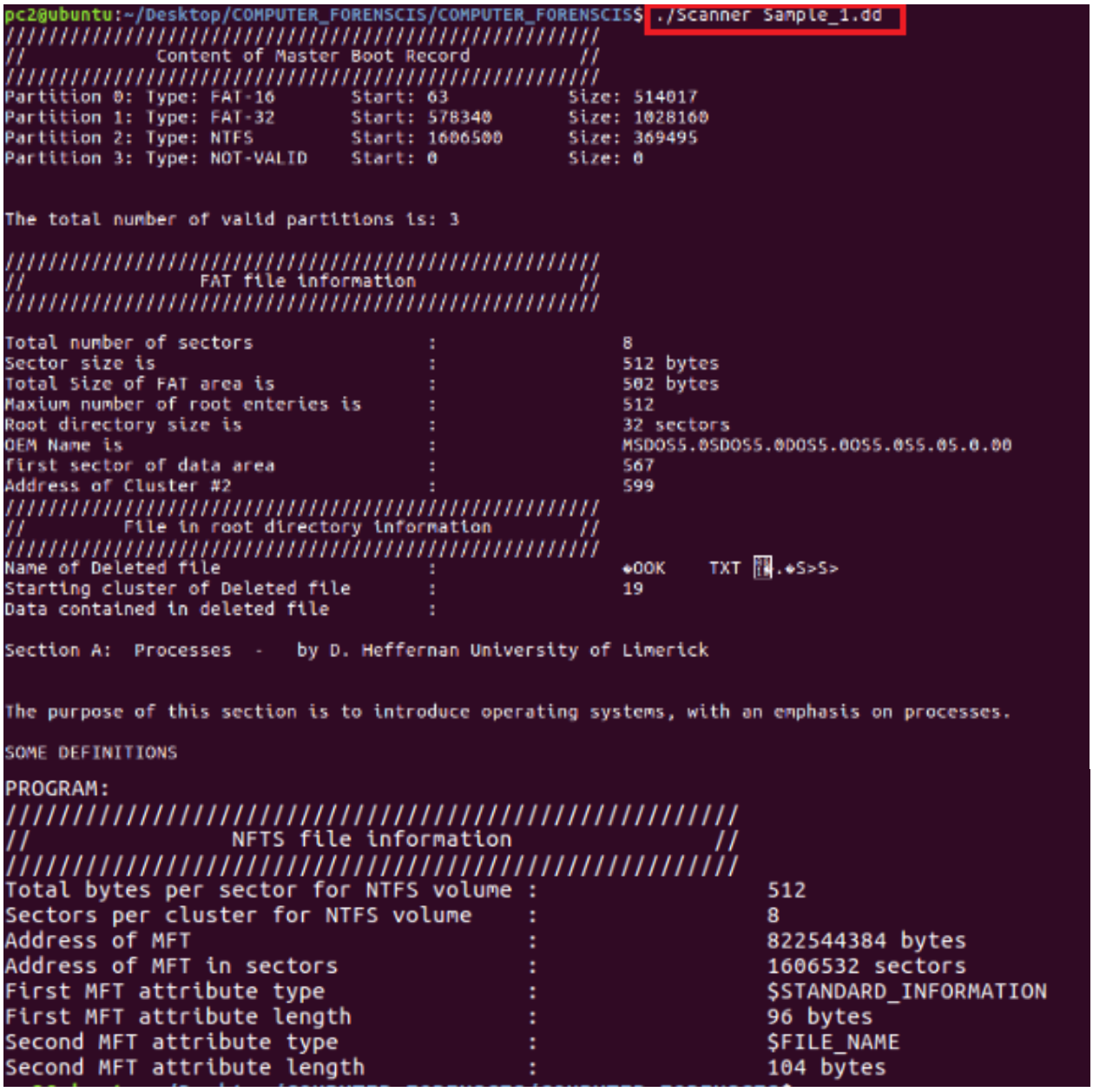


**Figure 4: Output of the forensic tool analysing disk image Sample_1.dd**

## Statement of completion:

This is a full report of Group 7 Phase 1 and Phase 2 (Full tool). Detailed instructions in accessing the source file available in GitHub was outlined and all copy of the source code are attached. A screen capture for a successful test in running the forensic tool against a disk image created using FTK Imager was included and shown in Figure 3.

For the final stage, the analysis of an NFTS Volume information was successfully completed and displayed in Figure 4.

Overall, Group 7 was able to complete the task required. Each member of the team work with their individual strength and collabrate effectively through team effort and communication.

This is a very interesting module in which we all gain knowledge and skills that we can apply out in the industry.

## Source Code:

### Scanner.h

```
/**
 * 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 pointer
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;

/**
 * struct NFTS FILEsystem
 */
struct NFTS
{
        int Num_sectors;
        int sectors_per_cluster;
        int cluster_number_of_MFT;
        int address_of_MFT;
        char first_MFT_attribute_type[24];
        int first_MFT_attribute_length;
        char second_MFT_attribute_type[24];
        int second_MFT_attribute_length;

} nfts;

/**
 * Function declarations;
 */
void getFatinfo();
char *concat(const char *s1, const char *s2);
void print_Header(char *s);
char *getOEM(const char *s1);
void getFileFromFatInfo();
void getNFTS_INFO();
void MFT_attribute_type(int type, char strtype[24]);
#endif
```

**MAIN FILE**

```
/**
 * 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();
```

```
        getNFTS_INFO();
        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);
        print_Header("      NFTS file information       ");
        printf("Total bytes per sector for NTFS volume :\t\t %d\n", nfts.Num_sectors);
        printf("Sectors per cluster for NTFS volume    :\t\t %d\n",
nfts.sectors_per_cluster);
        printf("Address of MFT                         :\t\t %d bytes\n",
nfts.address_of_MFT);
        printf("Address of MFT in sectors              :\t\t %d sectors\n",
nfts.address_of_MFT/512);
        printf("First MFT attribute type               :\t\t %s\n",
nfts.first_MFT_attribute_type);
        printf("First MFT attribute length             :\t\t %d bytes\n",
nfts.first_MFT_attribute_length);
        printf("Second MFT attribute type              :\t\t %s\n",
nfts.second_MFT_attribute_type);
        printf("Second MFT attribute length            :\t\t %d bytes\n",
nfts.second_MFT_attribute_length);

        fclose(fp);

        return (0);
}
```

**Scanner_part1b.c 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=malloc(strlen(root_buff+start_point_for_name)+1);
        DelFile.name=strcpy(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

      }
```

**Scanner_part1c.c File**

```
#include "Scanner.h"

/**
 * ########### REQUIEMENTS ################
 * - How many bytes per sector for this NTFS volume
 * - How many sectors per cluster for this NTFS volume
 * - What is the sector address for the $MFT file record
 * - What is the type and length of the first two attributes in the $MFT record
 * ########################################################
 */


void getNFTS_INFO(){

  char buf_part_table[100];
  int startsect = part_entry[2].start_sect * 512; //get starting piont of NTFS sector
  fseek(fp, startsect, SEEK_SET); // Seek to the start of the part_entry list
  fread(buf_part_table, 1, 100, fp);
  char temp_buff[2];
  for(int i=0x0B; i< 0x0D;i++){
    sprintf(temp_buff, "%d",*(buf_part_table +i));
  }
  nfts.Num_sectors = (atoi(temp_buff))<<8;
  nfts.sectors_per_cluster = *(buf_part_table+0x0D);

/**
 * code to get bit address of MFT
 */
  char $MFT_temp_buff[10];
  long int num=0;
  int temp;
  //gets the cluster number of MFT
  for(int i=0x37; i>= 0x30;i--){
      int count=0x30;
      $MFT_temp_buff[i-count]=*(buf_part_table +i);
      temp =(int)$MFT_temp_buff[i-count];
      num+=temp;
      if(i>0x30)
      num=num*10;

  }
nfts.cluster_number_of_MFT=num;
//address of MFT = (cluster_number_of_MFT * nfts.sectors_per_cluster *
nfts.sectors_per_cluster)
nfts.address_of_MFT= nfts.cluster_number_of_MFT * nfts.Num_sectors *
nfts.sectors_per_cluster +startsect;


//What is the type and length of the first two attributes in the $MFT record

char Standard_Information[400];
 fseek(fp, nfts.address_of_MFT, SEEK_SET); // Seek to the start $MFT
 fread(Standard_Information, 1, 400, fp);
 /*
 start of first MFT attribute
  */
 int first_MFT_attribute_offset= *(Standard_Information+0x14);
```

```
 //get Attribute type identifier of first attribute
 int Attribute_type_identifier= *(Standard_Information+first_MFT_attribute_offset);

 //copy its string value to NFTS structure
 MFT_attribute_type(Attribute_type_identifier,nfts.first_MFT_attribute_type);

 //get length of if offset got from Table N5: Data structure for a basic MFT record
 nfts.first_MFT_attribute_length=*(Standard_Information+first_MFT_attribute_offset+4);

 //first offset + first MFT attribute length = start of second
 int Second_MFT_attribute_offset
=first_MFT_attribute_offset+nfts.first_MFT_attribute_length;

 //get value at offset
 int Second_Attribute_type_identifier=
*(Standard_Information+Second_MFT_attribute_offset);
 MFT_attribute_type(Second_Attribute_type_identifier,nfts.second_MFT_attribute_type);
 nfts.second_MFT_attribute_length=*(Standard_Information+Second_MFT_attribute_offset+4);
}
```

**FUNCTIONS FILE**

```
#include "Scanner.h"

/**
 * Function to concatanate two string values
 * @param  s1 first string
 * @param  s2
 * @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");
}
/**
 * function to return the attribute type returned form MFT
 * @param type [description]
 */
void MFT_attribute_type(int type, char strtype[24]){
        switch (type)
        {
        case 0x10:
                strcpy(strtype, "$STANDARD_INFORMATION");
                break;
        case 0x20:
```

```
                strcpy(strtype, "$ATTRIBUTE_LIST");
                break;
        case 0x30:
                strcpy(strtype, "$FILE_NAME");
                break;
        case 0x40:
                strcpy(strtype, "$OBJECT_ID");
                break;
        case 0x60:
                strcpy(strtype, "$VOLUME_NAME");
                break;
        case 0x70:
                strcpy(strtype, "$VOLUME_INFORMATION");
                break;
        case 0x80:
                strcpy(strtype, "$DATA");
                break;
        case 0x90:
                strcpy(strtype, "$INDEX_ROOT");
                break;
        case 0xA0:
                strcpy(strtype, "$INDEX_ALLOCATION");
                break;
        case 0xB0:
                strcpy(strtype, "$BITMAP");
                break;
        case 0xC0:
                strcpy(strtype, "$REPARSE_POINT");
                break;
        default:
                strcpy(strtype, "NOT-DECODED CORRECTLY");
                break;
        }

}
```

**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 Scanner_part1c.o
```

```
	gcc -o Scanner ScannerMain.o functions.o Scanner_part1b.o Scanner_part1c.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 Scanner_part1c.c file
Scanner_part1c.o: Scanner_part1c.c
	gcc -Wall -ggdb -c Scanner_part1c.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
```