Mini Fat System

By Randa Adel Abd El Aziz and Reham Refaat Zaki.

The project implemented using C#.

The project has many classes work together:

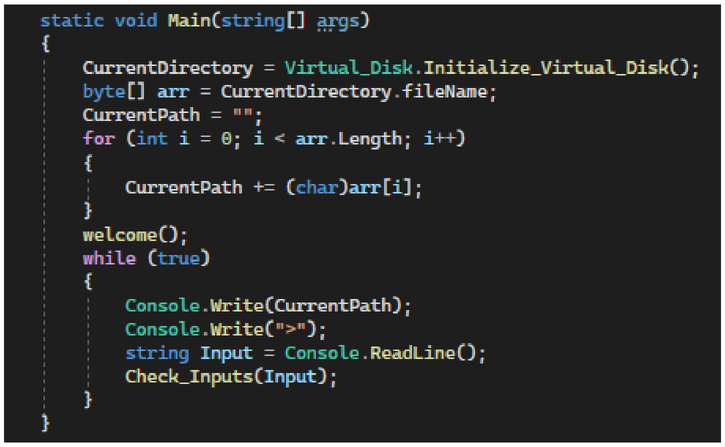
1. First Part Basic Classes.
2. Second Part Command Classes.

1. First Part Basic Classes:

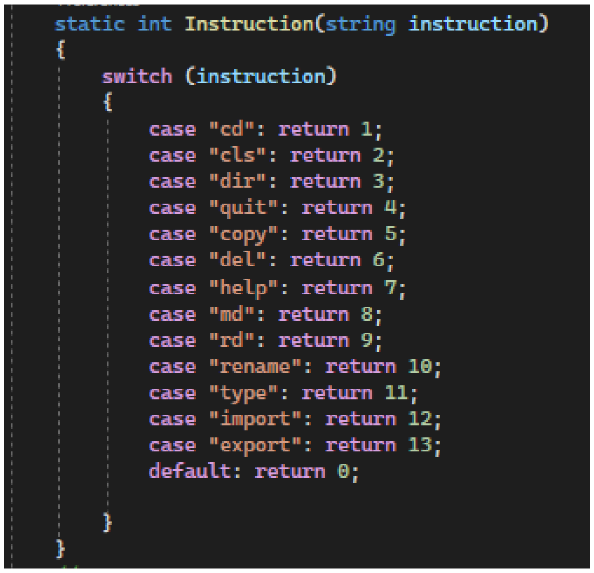
- **Program Class:**

It's the main class in project and from there we:

1) Create the root directory.
2) Input the command and it's arguments.
3) Check the validity of the command and it's arguments.
4) Call the appropriate class of that command to do it's function.

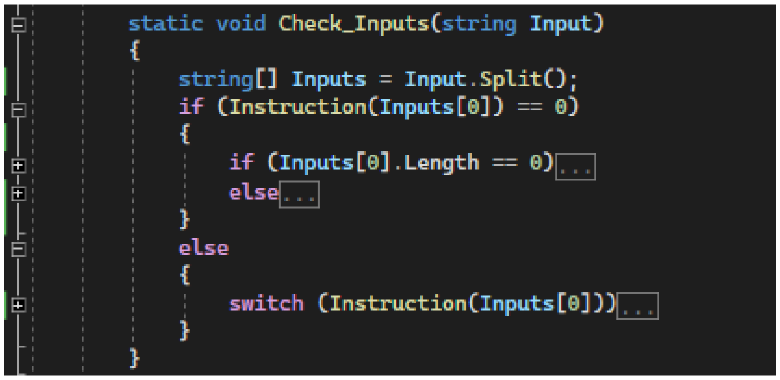


```
static void Main(string[] args)
{
    CurrentDirectory = Virtual_Disk.Initialize_Virtual_Disk();
    byte[] arr = CurrentDirectory.fileName;
    CurrentPath = "";
    for (int i = 0; i < arr.Length; i++)
    {
        CurrentPath += (char)arr[i];
    }
    welcome();
    while (true)
    {
        Console.Write(CurrentPath);
        Console.Write(">");
        string Input = Console.ReadLine();
        Check_Inputs(Input);
    }
}
```



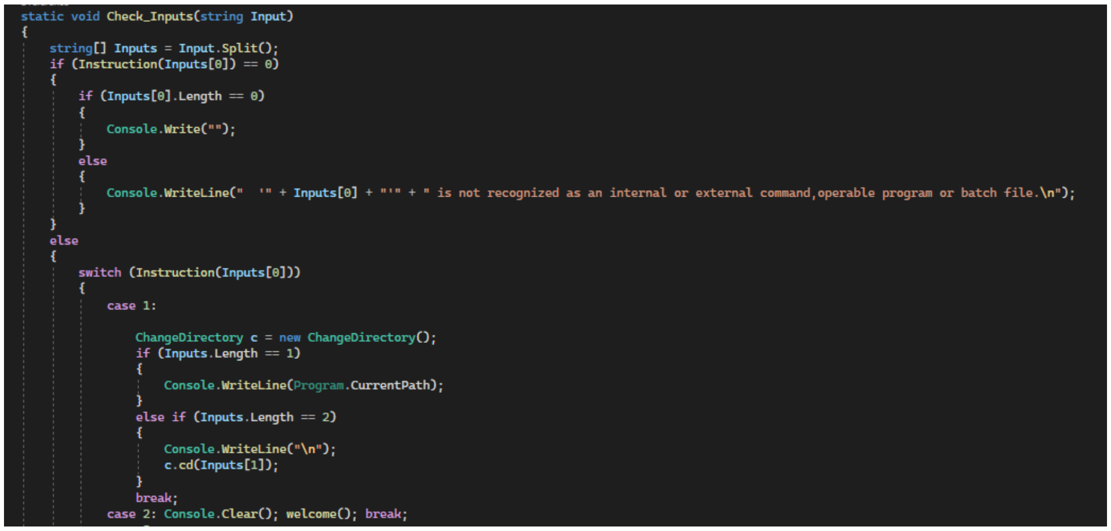
```
static int Instruction(string instruction)
{
    switch (instruction)
    {
        case "cd": return 1;
        case "cls": return 2;
        case "dir": return 3;
        case "quit": return 4;
        case "copy": return 5;
        case "del": return 6;
        case "help": return 7;
        case "md": return 8;
        case "rd": return 9;
        case "rename": return 10;
        case "type": return 11;
        case "import": return 12;
        case "export": return 13;
        default: return 0;

    }
}
```

From the function instruction we give a unique id for each command and use it to check the existence of that command.



```
static void Check_Inputs(string Input)
{
    string[] Inputs = Input.Split();
    if (Instruction(Inputs[0]) == 0)
    {
        if (Inputs[0].Length == 0)...
        else...
    }
    else
    {
        switch (Instruction(Inputs[0]))...
    }
}
```

In the function check input, we split the string that entered by the user and store it in array of string then use the index zero of the array as the command name and the next indices as arguments then we use the id given in function “instruction” to check the existence of that command and call the appropriate class of it.



```
static void Check_Inputs(string Input)
{
    string[] Inputs = Input.Split();
    if (Instruction(Inputs[0]) == 0)
    {
        if (Inputs[0].Length == 0)
        {
            Console.Write("");
        }
        else
        {
            Console.WriteLine("  '" + Inputs[0] + "'" + " is not recognized as an internal or external command,operable program or batch file.\n");
        }
    }
    else
    {
        switch (Instruction(Inputs[0]))
        {
            case 1:

                ChangeDirectory c = new ChangeDirectory();
                if (Inputs.Length == 1)
                {
                    Console.WriteLine(Program.CurrentPath);
                }
                else if (Inputs.Length == 2)
                {
                    Console.WriteLine("\n");
                    c.cd(Inputs[1]);
                }
                break;
            case 2: Console.Clear(); welcome(); break;
```

example : if user input space the length of the array would be zero so will give him a warning message but if the user for example “cd Desktop “ the array will store two indices first for “cd “ the second for “Desktop” then the id for this command is “1” so it will check if it exist in our commands or not then will make an instance of the class that change directory and check that it take it’s required arguments correct and then pass the arguments to the appropriate function in the class of change directory.

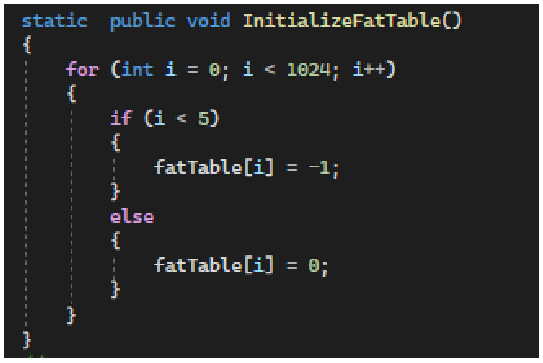
- **<u>Fat_Table Class:</u>**

In this class we create an array of 1024 to refer to the 1024 cluster of the virtual disk the first 5 indices we set to be -1 as they refer to the Fat_Table and in index 5 we store the root.

The class has many functions to handle the Fat array correctly:

1. **InitializeFatTable ():**

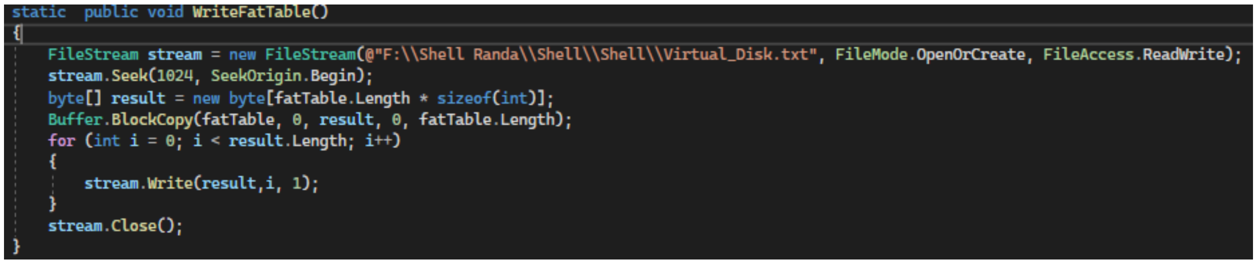
   Give initial value to the fat table -1 for the first 5 and the rest of the array index we give 0 to indicate that it's free to store in this cluster.



```
static  public void InitializeFatTable()
{
    for (int i = 0; i < 1024; i++)
    {
        if (i < 5)
        {
            fatTable[i] = -1;
        }
        else
        {
            fatTable[i] = 0;
        }
    }
}
```

2. **WriteFatTable():**

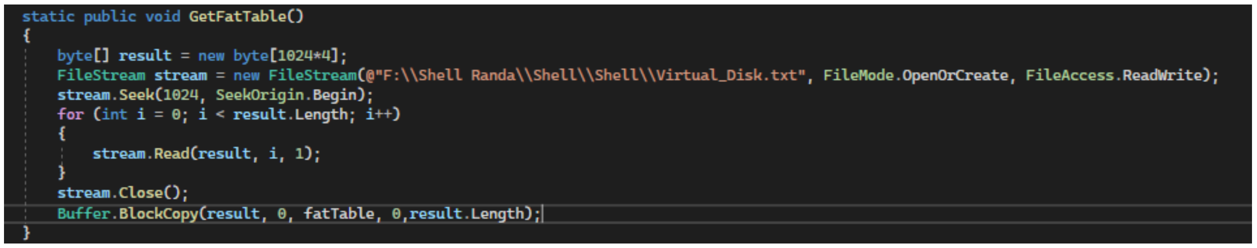
   We use this function to write only the fat table in the virtual Disk.



```
static  public void WriteFatTable()
{
    FileStream stream = new FileStream(@"F:\\Shell Randa\\Shell\\Shell\\Virtual_Disk.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite);
    stream.Seek(1024, SeekOrigin.Begin);
    byte[] result = new byte[fatTable.Length * sizeof(int)];
    Buffer.BlockCopy(fatTable, 0, result, 0, fatTable.Length);
    for (int i = 0; i < result.Length; i++)
    {
        stream.Write(result,i, 1);
    }
    stream.Close();
}
```

3. **GetFatTable():**

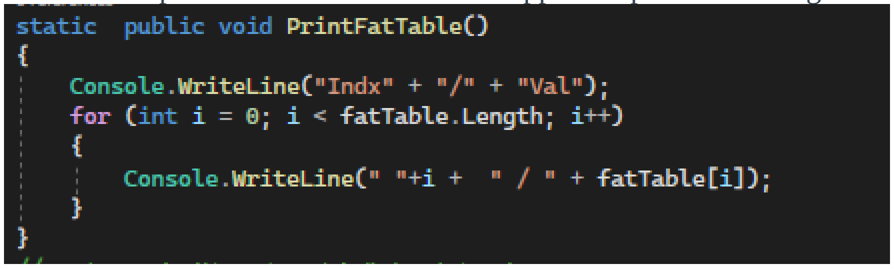
   We use it to read the Fat table from the virtual Disk.



```
static public void GetFatTable()
{
    byte[] result = new byte[1024*4];
    FileStream stream = new FileStream(@"F:\\Shell Randa\\Shell\\Shell\\Virtual_Disk.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite);
    stream.Seek(1024, SeekOrigin.Begin);
    for (int i = 0; i < result.Length; i++)
    {
        stream.Read(result, i, 1);
    }
    stream.Close();
    Buffer.BlockCopy(result, 0, fatTable, 0,result.Length);
}
```

4. **PrintFatTable()**

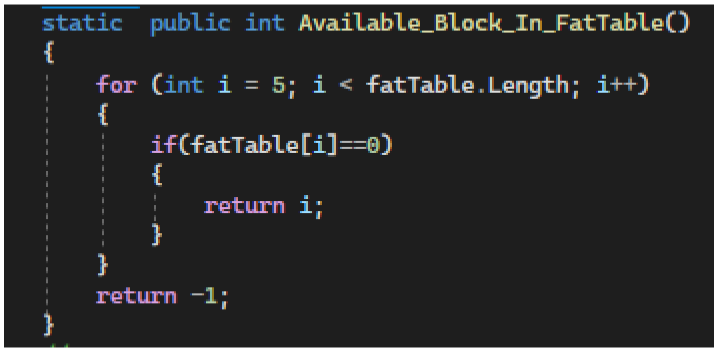
   We use it to print the fat table in console app to help us in tracking the values in fat table.



```
static  public void PrintFatTable()
{
    Console.WriteLine("Indx" + "/" + "Val");
    for (int i = 0; i < fatTable.Length; i++)
    {
        Console.WriteLine(" "+i +  " / " + fatTable[i]);
    }
}
```

5. **Available_Block_In_FatTable()**

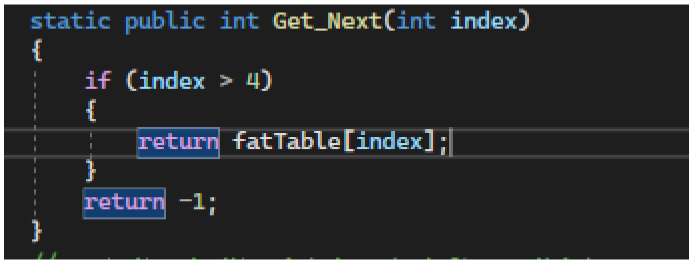
We use it to get the first available block that free to store in it.



```
static  public int Available_Block_In_FatTable()
{
    for (int i = 5; i < fatTable.Length; i++)
    {
        if(fatTable[i]==0)
        {
            return i;
        }
    }
    return -1;
}
```

6. **Get_Next(int index):**

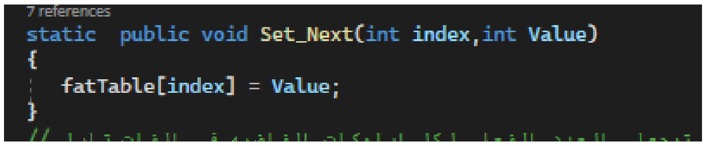
We use it in giving it the index of fat table and provide us with the value stored int it.



```
static public int Get_Next(int index)
{
    if (index > 4)
    {
        return fatTable[index];
    }
    return -1;
}
```

7. **Set_Next(int index,int Value)**

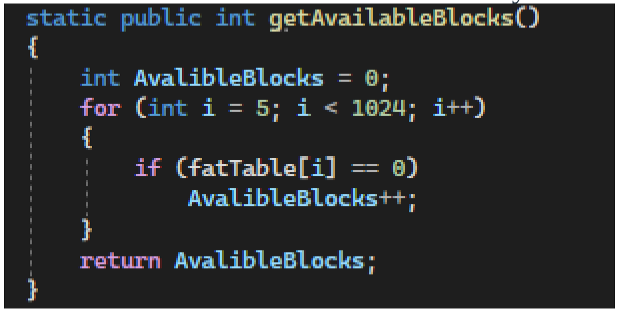
We use it to set appropriate value in to appropriate index.



```
7 references
static  public void Set_Next(int index,int Value)
{
    fatTable[index] = Value;
}
```

8. **getAvailableBlocks()**

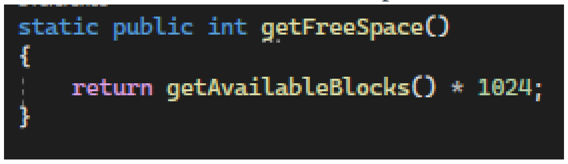
we use it to count the total available block in system.



```
static public int getAvailableBlocks()
{
    int AvalibleBlocks = 0;
    for (int i = 5; i < 1024; i++)
    {
        if (fatTable[i] == 0)
            AvalibleBlocks++;
    }
    return AvalibleBlocks;
}
```

9. **getFreeSpace()**

   we use it to calculate the free space of the virtual disk.



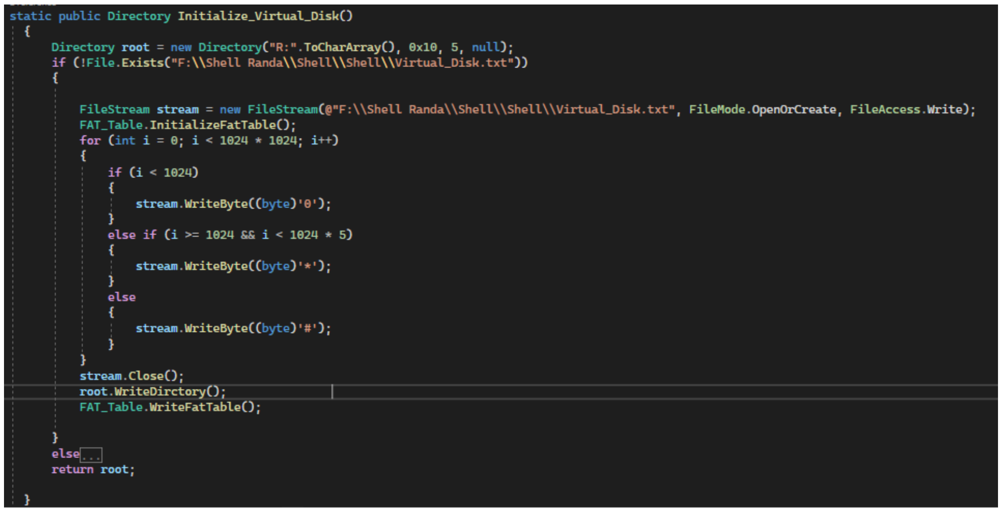
```
static public int getFreeSpace()
{
    return getAvailableBlocks() * 1024;
}
```

- **<u>Virtual Disk Class:</u>**

  This class is creating a virtual disk in our computer by making file of size 1 Mega consist of 1024 cluster each cluster of size 1024 byte **we make this function in that class:**

1) **Initialize_Virtual_Disk();**

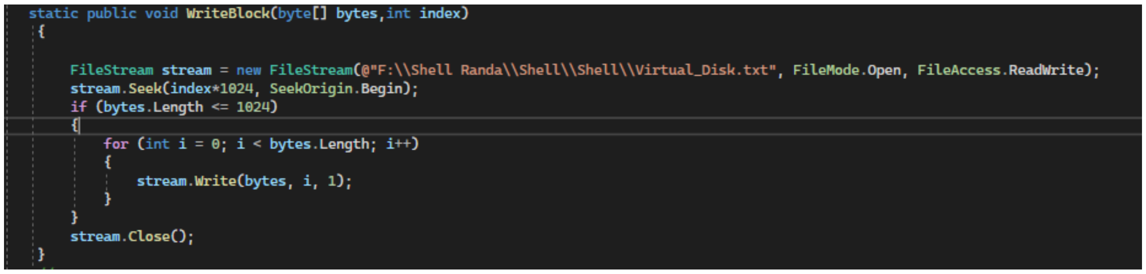
   This function responsible for making the virtual disk and the root directory as it is the first function to call in the program it initializes the virtual disk by storing the appropriate size for it and return the root directory.



```
static public Directory Initialize_Virtual_Disk()
{
    Directory root = new Directory("R:".ToCharArray(), 0x10, 5, null);
    if (!File.Exists("F:\\Shell Randa\\Shell\\Shell\\Virtual_Disk.txt"))
    {

        FileStream stream = new FileStream(@"F:\\Shell Randa\\Shell\\Shell\\Virtual_Disk.txt", FileMode.OpenOrCreate, FileAccess.Write);
        FAT_Table.InitializeFatTable();
        for (int i = 0; i < 1024 * 1024; i++)
        {
            if (i < 1024)
            {
                stream.WriteByte((byte)'0');
            }
            else if (i >= 1024 && i < 1024 * 5)
            {
                stream.WriteByte((byte)'*');
            }
            else
            {
                stream.WriteByte((byte)'#');
            }
        }
        stream.Close();
        root.WriteDirctory();
        FAT_Table.WriteFatTable();

    }
    else...
    return root;

}
```

2) **WriteBlock(byte[] bytes,int index):**

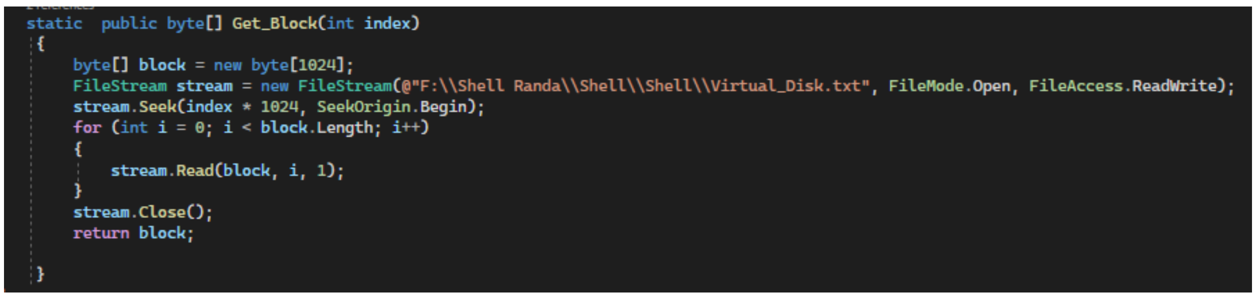
   This function takes a block and the number of the cluster that it writes it on it then write it in virtual disk.



```
static public void WriteBlock(byte[] bytes,int index)
{

    FileStream stream = new FileStream(@"F:\\Shell Randa\\Shell\\Shell\\Virtual_Disk.txt", FileMode.Open, FileAccess.ReadWrite);
    stream.Seek(index*1024, SeekOrigin.Begin);
    if (bytes.Length <= 1024)
    {
        for (int i = 0; i < bytes.Length; i++)
        {
            stream.Write(bytes, i, 1);
        }
    }
    stream.Close();
}
```

3) byte[] Get_Block(int index):

it read appropriate block referenced by its index and return it in array of bytes.

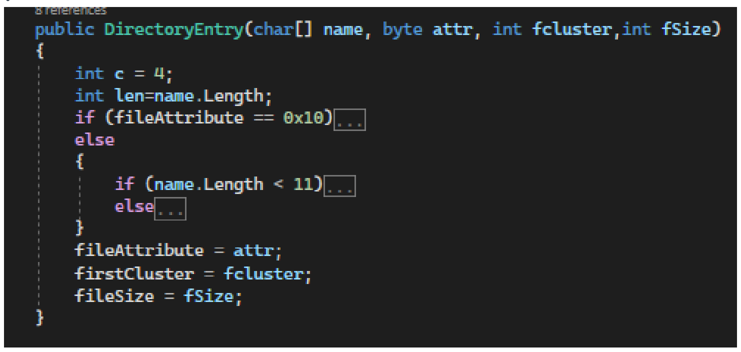


```
static  public byte[] Get_Block(int index)
{
    byte[] block = new byte[1024];
    FileStream stream = new FileStream(@"F:\\Shell Randa\\Shell\\Shell\\Virtual_Disk.txt", FileMode.Open, FileAccess.ReadWrite);
    stream.Seek(index * 1024, SeekOrigin.Begin);
    for (int i = 0; i < block.Length; i++)
    {
        stream.Read(block, i, 1);
    }
    stream.Close();
    return block;

}
```

- **<u>Class DirectoryEntry :</u>**

  This class responsible for creating an entry of the Directory table which could by file or directory first we initialize these values from the constructor of this class

  .

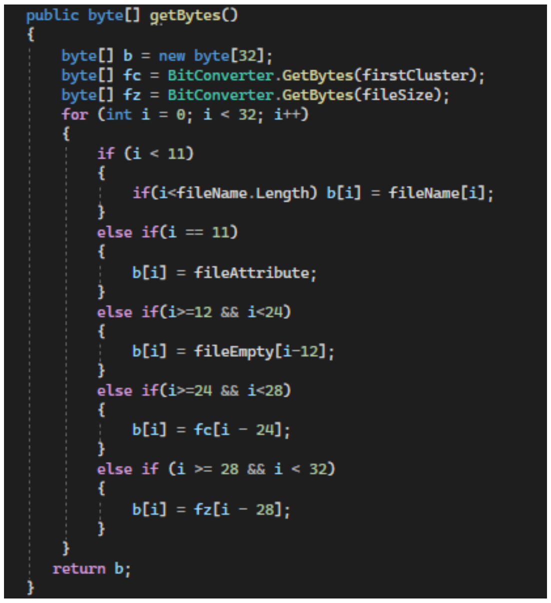


```
8 references
public DirectoryEntry(char[] name, byte attr, int fcluster,int fSize)
{
    int c = 4;
    int len=name.Length;
    if (fileAttribute == 0x10)...
    else
    {
        if (name.Length < 11)...
        else...
    }
    fileAttribute = attr;
    firstCluster = fcluster;
    fileSize = fSize;
}
```

It contains the following functions:

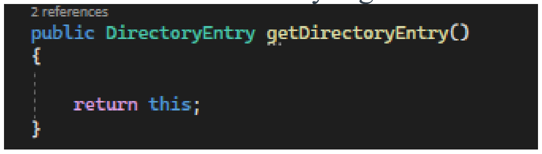
1) getBytes():

it returns the directory entry in form of array of bytes.

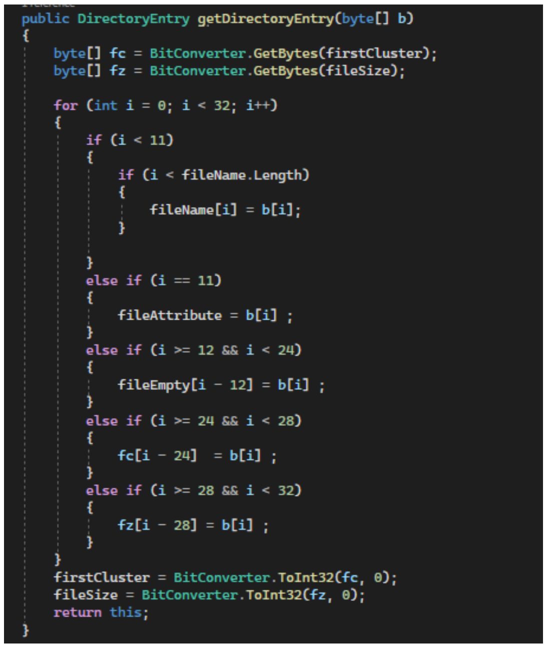


```
public byte[] getBytes()
{
    byte[] b = new byte[32];
    byte[] fc = BitConverter.GetBytes(firstCluster);
    byte[] fz = BitConverter.GetBytes(fileSize);
    for (int i = 0; i < 32; i++)
    {
        if (i < 11)
        {
            if(i<fileName.Length) b[i] = fileName[i];
        }
        else if(i == 11)
        {
            b[i] = fileAttribute;
        }
        else if(i>=12 && i<24)
        {
            b[i] = fileEmpty[i-12];
        }
        else if(i>=24 && i<28)
        {
            b[i] = fc[i - 24];
        }
        else if (i >= 28 && i < 32)
        {
            b[i] = fz[i - 28];
        }
    }
   return b;
}
```

2) **getDirectoryEntry(byte[] b):**
we override this function in two versions one to take as argument an array of bytes and store it in directory entry and the other version doesn't take any arguments and return a directory entry.



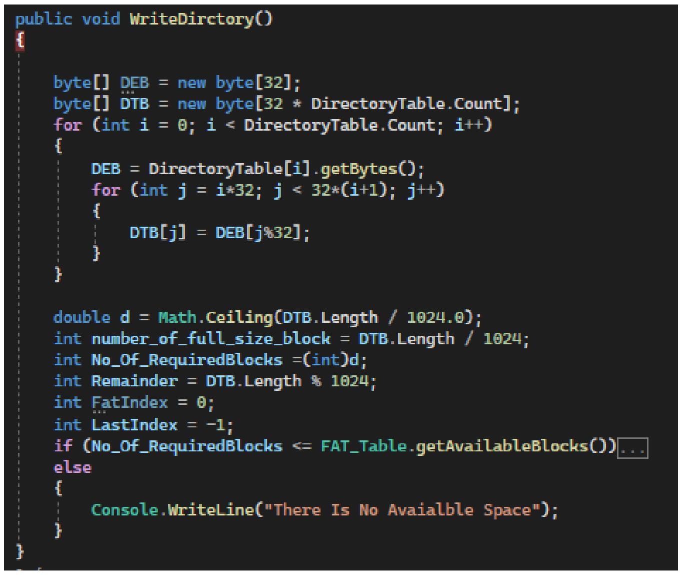
```
2 references
public DirectoryEntry getDirectoryEntry()
{

    return this;
}
```



```
public DirectoryEntry getDirectoryEntry(byte[] b)
{
    byte[] fc = BitConverter.GetBytes(firstCluster);
    byte[] fz = BitConverter.GetBytes(fileSize);

    for (int i = 0; i < 32; i++)
    {
        if (i < 11)
        {
            if (i < fileName.Length)
            {
                fileName[i] = b[i];
            }

        }
        else if (i == 11)
        {
            fileAttribute = b[i] ;
        }
        else if (i >= 12 && i < 24)
        {
            fileEmpty[i - 12] = b[i] ;
        }
        else if (i >= 24 && i < 28)
        {
            fc[i - 24]  = b[i] ;
        }
        else if (i >= 28 && i < 32)
        {
            fz[i - 28] = b[i] ;
        }
    }
    firstCluster = BitConverter.ToInt32(fc, 0);
    fileSize = BitConverter.ToInt32(fz, 0);
    return this;
}
```

- **<u>Directory Class:</u>**
  This class inherit from DirectorEntry class and make a directory table which store any directory or file would be make, **it contains the following functions:**

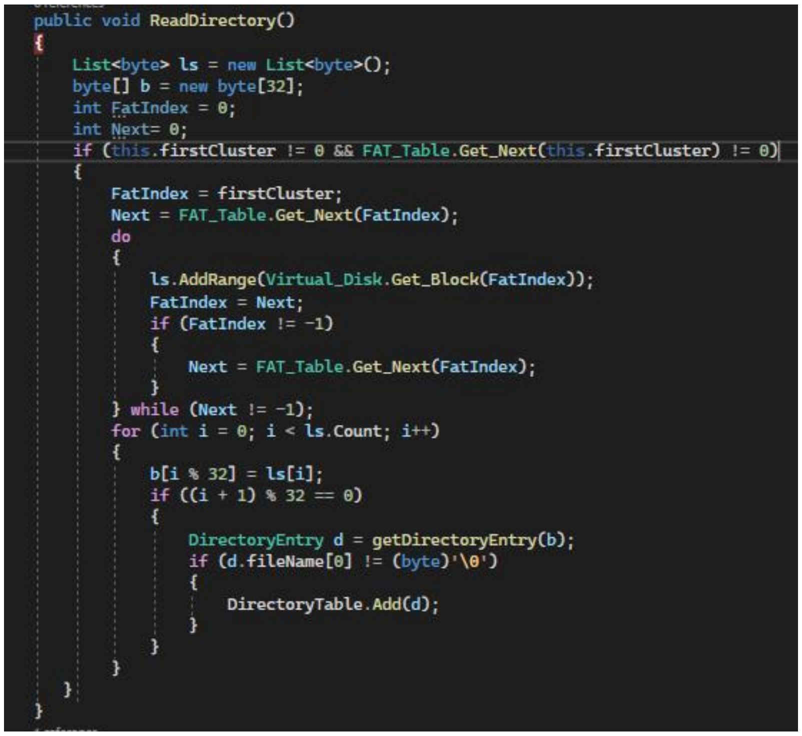
1) **WriteDirctory():**
It writes directory by checking the available space to store It then write it in a virtual disk.



```
public void WriteDirctory()
{

    byte[] DEB = new byte[32];
    byte[] DTB = new byte[32 * DirectoryTable.Count];
    for (int i = 0; i < DirectoryTable.Count; i++)
    {
        DEB = DirectoryTable[i].getBytes();
        for (int j = i*32; j < 32*(i+1); j++)
        {
            DTB[j] = DEB[j%32];
        }
    }

    double d = Math.Ceiling(DTB.Length / 1024.0);
    int number_of_full_size_block = DTB.Length / 1024;
    int No_Of_RequiredBlocks =(int)d;
    int Remainder = DTB.Length % 1024;
    int FatIndex = 0;
    int LastIndex = -1;
    if (No_Of_RequiredBlocks <= FAT_Table.getAvailableBlocks())...
    else
    {
        Console.WriteLine("There Is No Avaialble Space");
    }
}
```

2) ReadDirectory():

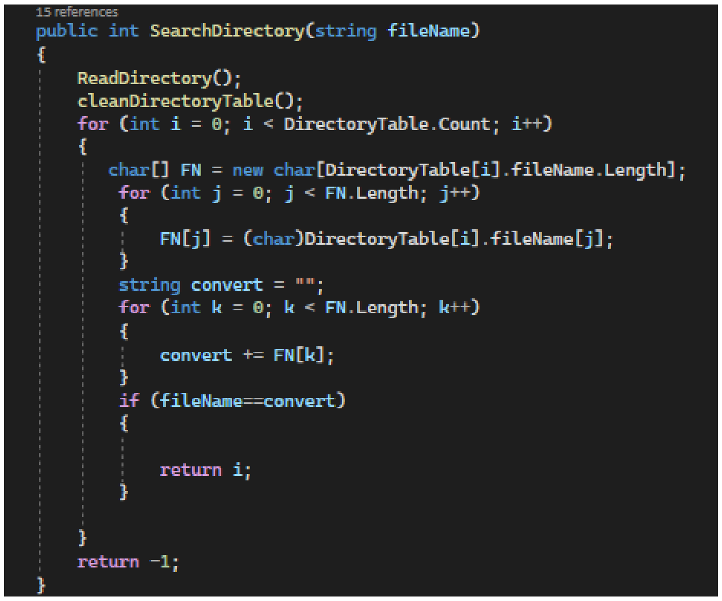
Responsible to write the directory from a virtual disk and store it again in directory table.



```
public void ReadDirectory()
{
    List<byte> ls = new List<byte>();
    byte[] b = new byte[32];
    int FatIndex = 0;
    int Next= 0;
    if (this.firstCluster != 0 && FAT_Table.Get_Next(this.firstCluster) != 0)
    {
        FatIndex = firstCluster;
        Next = FAT_Table.Get_Next(FatIndex);
        do
        {
            ls.AddRange(Virtual_Disk.Get_Block(FatIndex));
            FatIndex = Next;
            if (FatIndex != -1)
            {
                Next = FAT_Table.Get_Next(FatIndex);
            }
        } while (Next != -1);
        for (int i = 0; i < ls.Count; i++)
        {
            b[i % 32] = ls[i];
            if ((i + 1) % 32 == 0)
            {
                DirectoryEntry d = getDirectoryEntry(b);
                if (d.fileName[0] != (byte)'\0')
                {
                    DirectoryTable.Add(d);
                }
            }
        }
    }
}
```

3) SearchDirectory(string fileName):

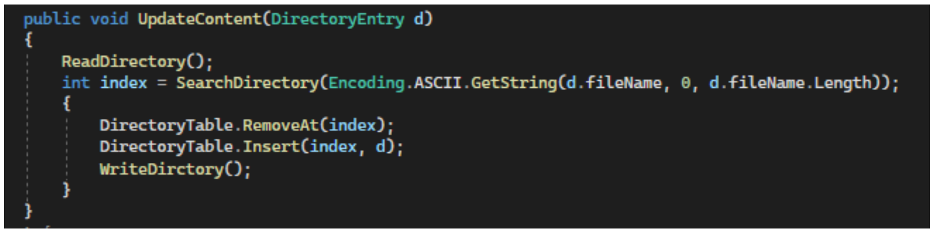
It takes a file name and chick the existence of that file by searching for it in directory table.



```
15 references
public int SearchDirectory(string fileName)
{
    ReadDirectory();
    cleanDirectoryTable();
    for (int i = 0; i < DirectoryTable.Count; i++)
    {
        char[] FN = new char[DirectoryTable[i].fileName.Length];
        for (int j = 0; j < FN.Length; j++)
        {
            FN[j] = (char)DirectoryTable[i].fileName[j];
        }
        string convert = "";
        for (int k = 0; k < FN.Length; k++)
        {
            convert += FN[k];
        }
        if (fileName==convert)
        {

            return i;
        }

    }
    return -1;
}
```

4) UpdateContent(DirectoryEntry d):

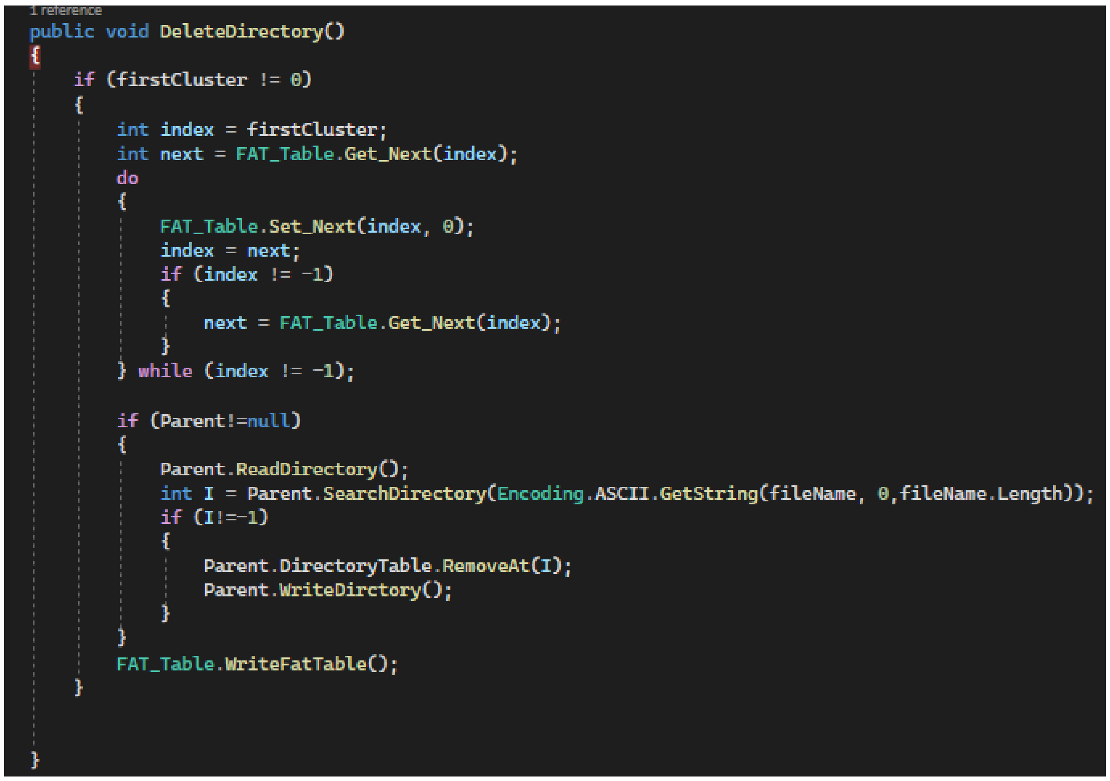
It updates the record in directory table by searching the existence of that record then take that index and delete it and store the new record as an exchangeable of it.



```
public void UpdateContent(DirectoryEntry d)
{
    ReadDirectory();
    int index = SearchDirectory(Encoding.ASCII.GetString(d.fileName, 0, d.fileName.Length));
    {
        DirectoryTable.RemoveAt(index);
        DirectoryTable.Insert(index, d);
        WriteDirctory();
    }
}
```

5) DeleteDirectory():

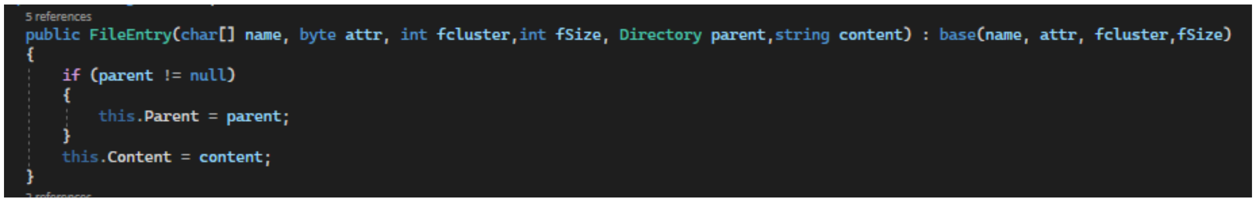
It deletes a record of directory entry by checking if it’s a subdirectory of any directory and delete it.



```
1 reference
public void DeleteDirectory()
{
    if (firstCluster != 0)
    {
        int index = firstCluster;
        int next = FAT_Table.Get_Next(index);
        do
        {
            FAT_Table.Set_Next(index, 0);
            index = next;
            if (index != -1)
            {
                next = FAT_Table.Get_Next(index);
            }
        } while (index != -1);

        if (Parent!=null)
        {
            Parent.ReadDirectory();
            int I = Parent.SearchDirectory(Encoding.ASCII.GetString(fileName, 0,fileName.Length));
            if (I!=-1)
            {
                Parent.DirectoryTable.RemoveAt(I);
                Parent.WriteDirctory();
            }
        }
        FAT_Table.WriteFatTable();
    }


}
```

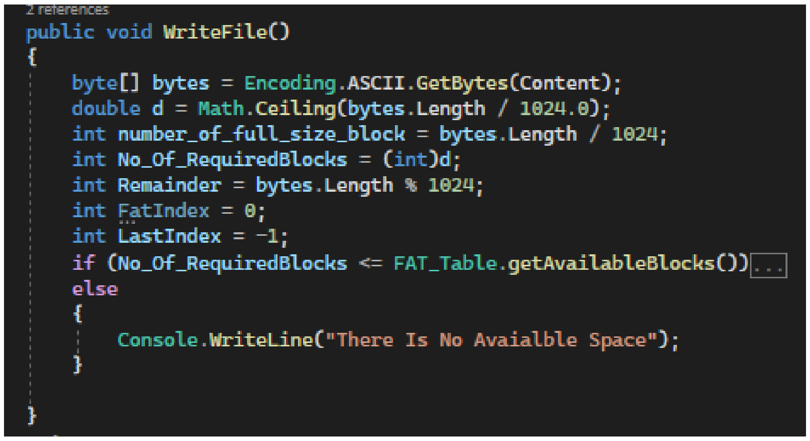
- **FileEntery Class:**

  This class responsible to handle files and it’s content and inherits from class directory entry it has a constructor to initialize this value and contains the following functions.



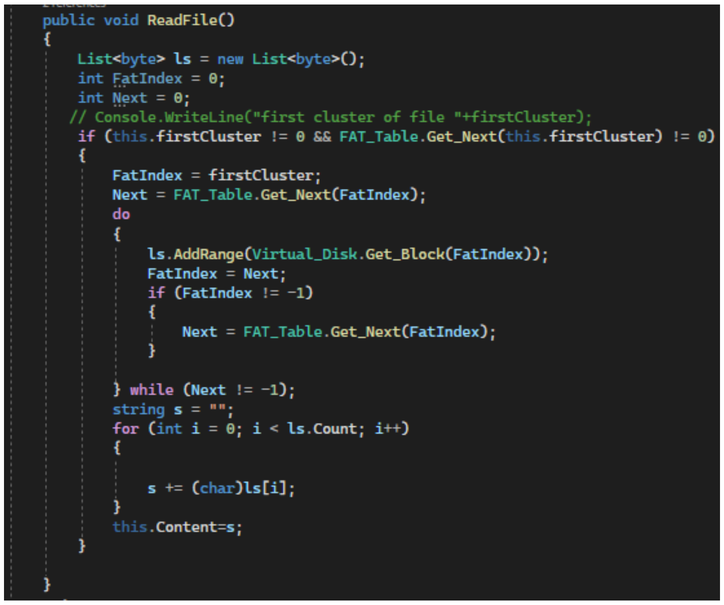
```
5 references
public FileEntry(char[] name, byte attr, int fcluster,int fSize, Directory parent,string content) : base(name, attr, fcluster,fSize)
{
    if (parent != null)
    {
        this.Parent = parent;
    }
    this.Content = content;
}
```

1) **WriteFile():**
   This function responsible to write the file and its content in a virtual disk.



```
2 references
public void WriteFile()
{
    byte[] bytes = Encoding.ASCII.GetBytes(Content);
    double d = Math.Ceiling(bytes.Length / 1024.0);
    int number_of_full_size_block = bytes.Length / 1024;
    int No_Of_RequiredBlocks = (int)d;
    int Remainder = bytes.Length % 1024;
    int FatIndex = 0;
    int LastIndex = -1;
    if (No_Of_RequiredBlocks <= FAT_Table.getAvailableBlocks())...
    else
    {
        Console.WriteLine("There Is No Avaialble Space");
    }

}
```

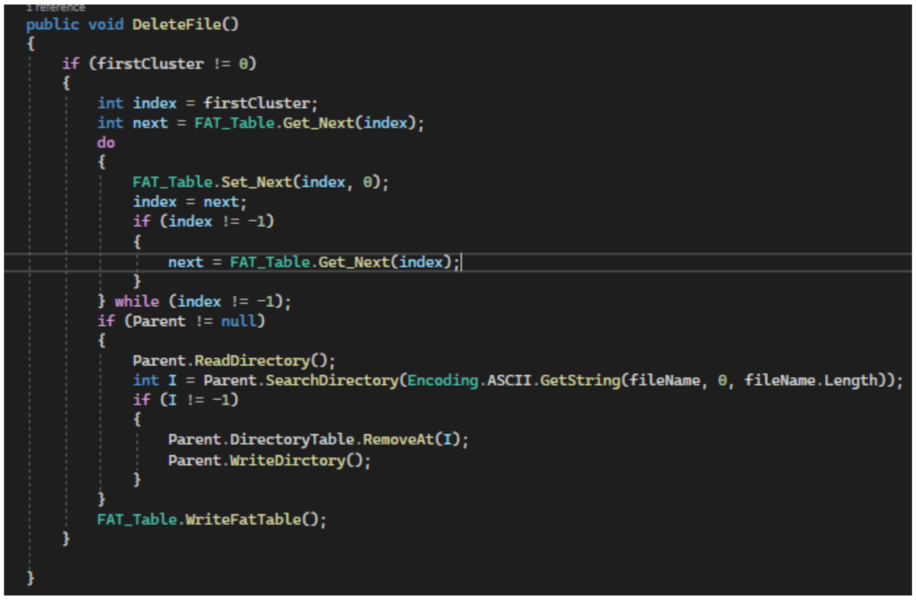
2) **ReadFile():**
   This function read the file from virtual disk and return the content of the file.



```
public void ReadFile()
{
    List<byte> ls = new List<byte>();
    int FatIndex = 0;
    int Next = 0;
   // Console.WriteLine("first cluster of file "+firstCluster);
    if (this.firstCluster != 0 && FAT_Table.Get_Next(this.firstCluster) != 0)
    {
        FatIndex = firstCluster;
        Next = FAT_Table.Get_Next(FatIndex);
        do
        {
            ls.AddRange(Virtual_Disk.Get_Block(FatIndex));
            FatIndex = Next;
            if (FatIndex != -1)
            {
                Next = FAT_Table.Get_Next(FatIndex);
            }

        } while (Next != -1);
        string s = "";
        for (int i = 0; i < ls.Count; i++)
        {

            s += (char)ls[i];
        }
        this.Content=s;
    }

}
```

3) DeleteFile():

This function deletes the file by remove it from directory entry.



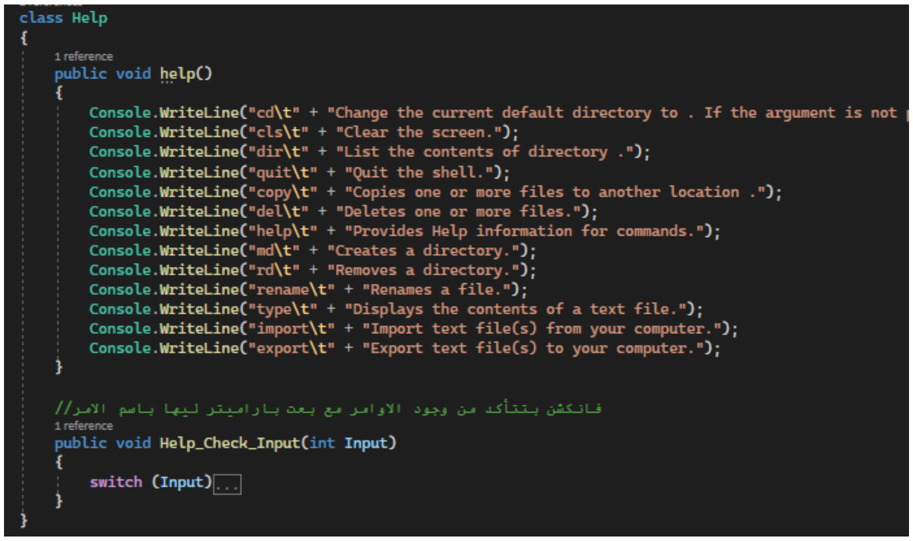
```
public void DeleteFile()
{
    if (firstCluster != 0)
    {
        int index = firstCluster;
        int next = FAT_Table.Get_Next(index);
        do
        {
            FAT_Table.Set_Next(index, 0);
            index = next;
            if (index != -1)
            {
                next = FAT_Table.Get_Next(index);
            }
        } while (index != -1);
        if (Parent != null)
        {
            Parent.ReadDirectory();
            int I = Parent.SearchDirectory(Encoding.ASCII.GetString(fileName, 0, fileName.Length));
            if (I != -1)
            {
                Parent.DirectoryTable.RemoveAt(I);
                Parent.WriteDirctory();
            }
        }
        FAT_Table.WriteFatTable();
    }

}
```

2. Second Part Command Classes:

- **<u>Help Class:</u>**

It has to versions first when user input the command “help” only and the second version when enter help with other command example “help md”.

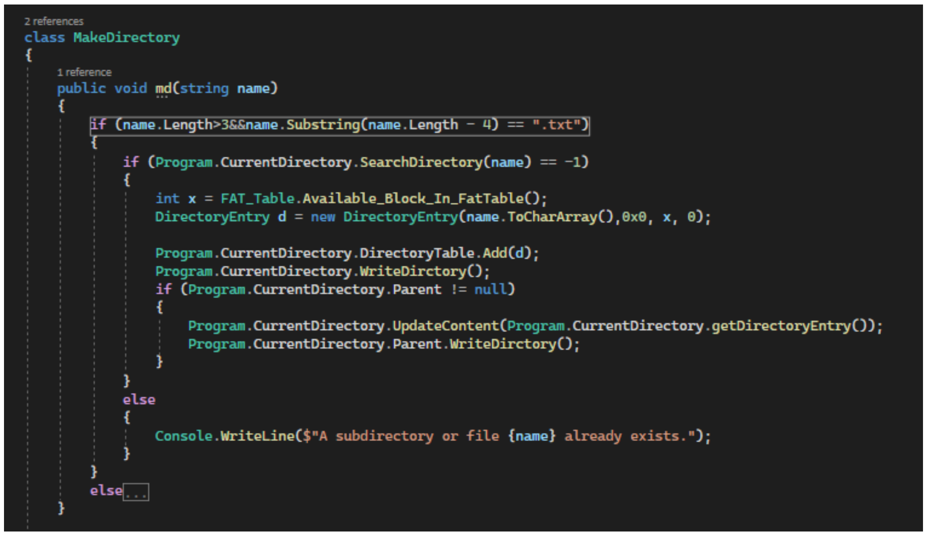
So, we make to function to manipulate the two cases:



```
class Help
{
    1 reference
    public void help()
    {
        Console.WriteLine("cd\t" + "Change the current default directory to . If the argument is not
        Console.WriteLine("cls\t" + "Clear the screen.");
        Console.WriteLine("dir\t" + "List the contents of directory .");
        Console.WriteLine("quit\t" + "Quit the shell.");
        Console.WriteLine("copy\t" + "Copies one or more files to another location .");
        Console.WriteLine("del\t" + "Deletes one or more files.");
        Console.WriteLine("help\t" + "Provides Help information for commands.");
        Console.WriteLine("md\t" + "Creates a directory.");
        Console.WriteLine("rd\t" + "Removes a directory.");
        Console.WriteLine("rename\t" + "Renames a file.");
        Console.WriteLine("type\t" + "Displays the contents of a text file.");
        Console.WriteLine("import\t" + "Import text file(s) from your computer.");
        Console.WriteLine("export\t" + "Export text file(s) to your computer.");
    }

    //فانكشن بتتأكد من وجود الاوامر مع بعت باراميتر ليها باسم الامر
    1 reference
    public void Help_Check_Input(int Input)
    {
        switch (Input)...
    }
}
```

- **MakeDirectory Class:**

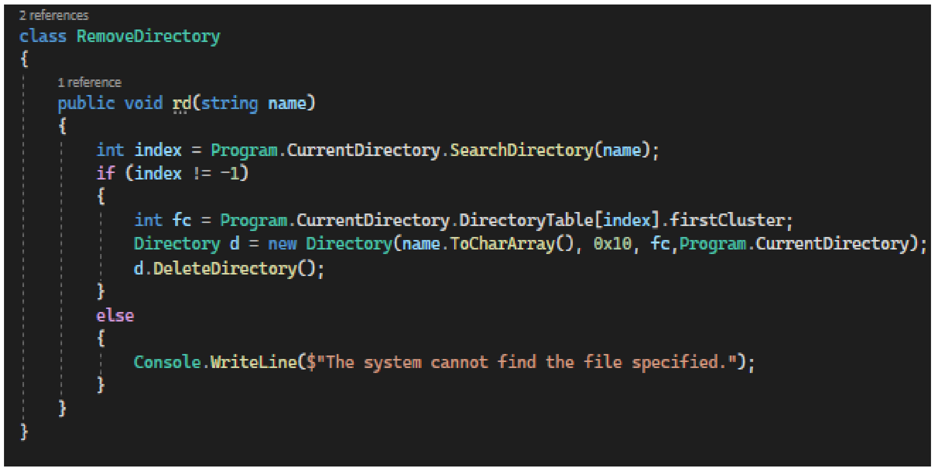
It responsible for make new directory by checking the existence of the directory and make instance of directory entry and add it to directory table



```
2 references
class MakeDirectory
{
    1 reference
    public void md(string name)
    {
        if (name.Length>3&&name.Substring(name.Length - 4) == ".txt")
        {
            if (Program.CurrentDirectory.SearchDirectory(name) == -1)
            {
                int x = FAT_Table.Available_Block_In_FatTable();
                DirectoryEntry d = new DirectoryEntry(name.ToCharArray(),0x0, x, 0);

                Program.CurrentDirectory.DirectoryTable.Add(d);
                Program.CurrentDirectory.WriteDirctory();
                if (Program.CurrentDirectory.Parent != null)
                {
                    Program.CurrentDirectory.UpdateContent(Program.CurrentDirectory.getDirectoryEntry());
                    Program.CurrentDirectory.Parent.WriteDirctory();
                }
            }
            else
            {
                Console.WriteLine($"A subdirectory or file {name} already exists.");
            }
        }
        else...
    }
```

- **RemoveDirectory Class:**

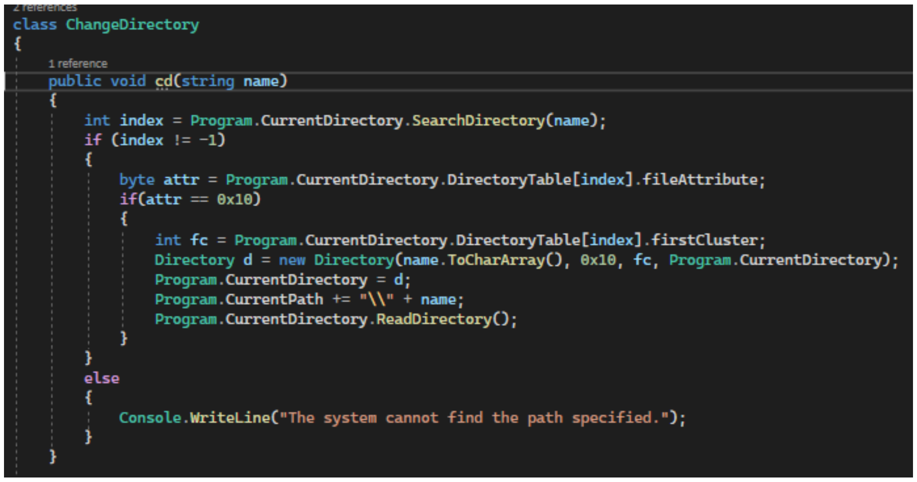
This class delete a specify directory by searching for it and use function delete in directory class.



```
2 references
class RemoveDirectory
{
    1 reference
    public void rd(string name)
    {
        int index = Program.CurrentDirectory.SearchDirectory(name);
        if (index != -1)
        {
            int fc = Program.CurrentDirectory.DirectoryTable[index].firstCluster;
            Directory d = new Directory(name.ToCharArray(), 0x10, fc,Program.CurrentDirectory);
            d.DeleteDirectory();
        }
        else
        {
            Console.WriteLine($"The system cannot find the file specified.");
        }
    }
}
```

- **<u>ChangeDirectory Class:</u>**

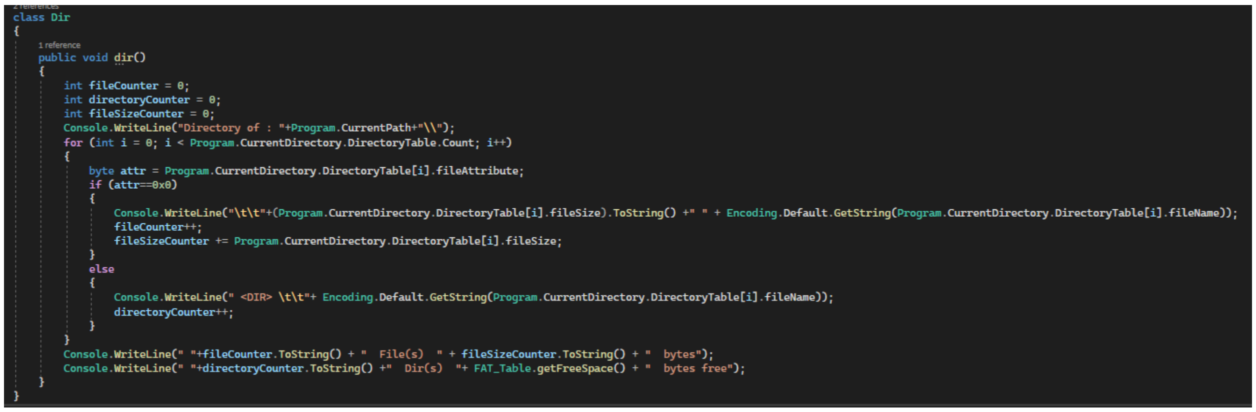
  It changes the current path to any existence path so we first search for the validity of the path where we change to it then we change the current directory.



```
2 references
class ChangeDirectory
{
    1 reference
    public void cd(string name)
    {
        int index = Program.CurrentDirectory.SearchDirectory(name);
        if (index != -1)
        {
            byte attr = Program.CurrentDirectory.DirectoryTable[index].fileAttribute;
            if(attr == 0x10)
            {
                int fc = Program.CurrentDirectory.DirectoryTable[index].firstCluster;
                Directory d = new Directory(name.ToCharArray(), 0x10, fc, Program.CurrentDirectory);
                Program.CurrentDirectory = d;
                Program.CurrentPath += "\\" + name;
                Program.CurrentDirectory.ReadDirectory();
            }
        }
        else
        {
            Console.WriteLine("The system cannot find the path specified.");
        }
    }
```

- **<u>Dir Class:</u>**

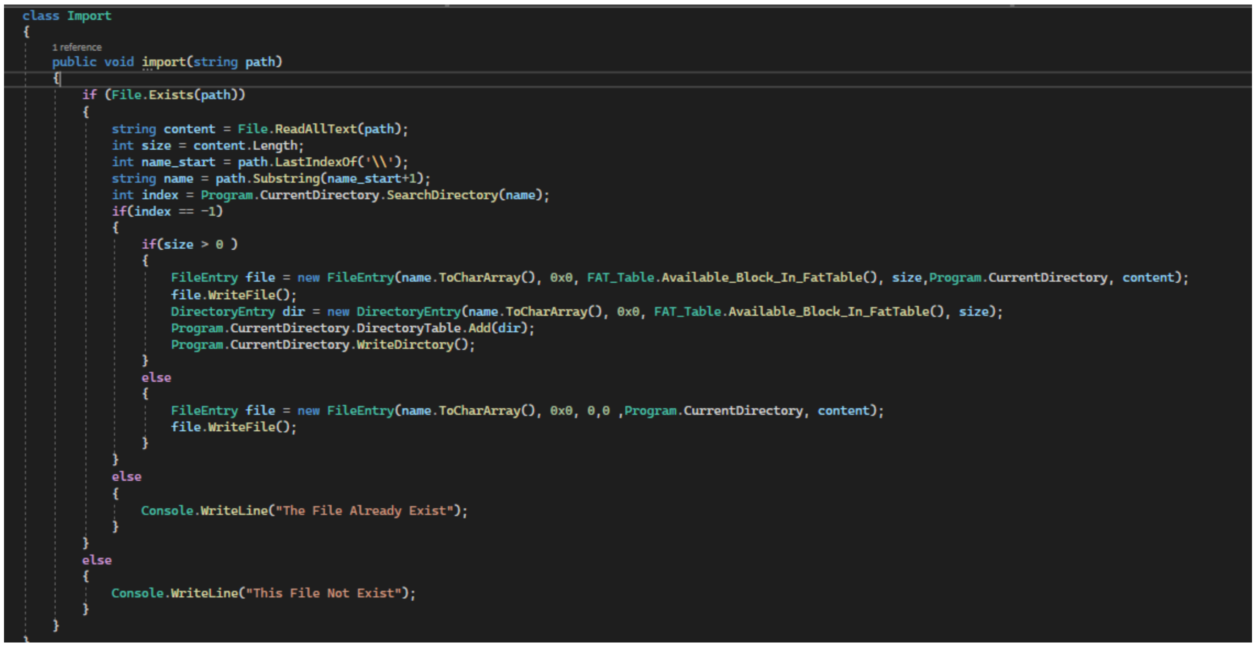
This class list the content of files and folders that exist in current directory.



```
2 references
class Dir
{
    1 reference
    public void dir()
    {
        int fileCounter = 0;
        int directoryCounter = 0;
        int fileSizeCounter = 0;
        Console.WriteLine("Directory of : "+Program.CurrentPath+"\\");
        for (int i = 0; i < Program.CurrentDirectory.DirectoryTable.Count; i++)
        {
            byte attr = Program.CurrentDirectory.DirectoryTable[i].fileAttribute;
            if (attr==0x0)
            {
                Console.WriteLine("\t\t"+(Program.CurrentDirectory.DirectoryTable[i].fileSize).ToString() +" " + Encoding.Default.GetString(Program.CurrentDirectory.DirectoryTable[i].fileName));
                fileCounter++;
                fileSizeCounter += Program.CurrentDirectory.DirectoryTable[i].fileSize;
            }
            else
            {
                Console.WriteLine(" <DIR> \t\t"+ Encoding.Default.GetString(Program.CurrentDirectory.DirectoryTable[i].fileName));
                directoryCounter++;
            }
        }
        Console.WriteLine(" "+fileCounter.ToString() + "  File(s)  " + fileSizeCounter.ToString() + "  bytes");
        Console.WriteLine(" "+directoryCounter.ToString() +"  Dir(s)  "+ FAT_Table.getFreeSpace() + "  bytes free");
    }
}
```

- **Import Class:**

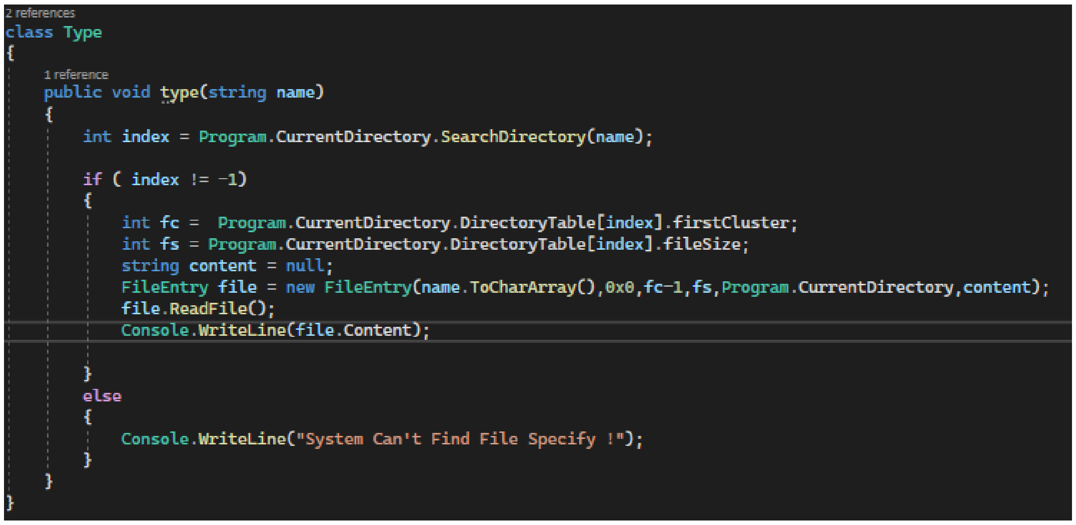
It responsible to import class from computer and store it with its content in virtual disk so it's chick the validity of the path in computer and import.



```
class Import
{
    1 reference
    public void import(string path)
    {
        if (File.Exists(path))
        {
            string content = File.ReadAllText(path);
            int size = content.Length;
            int name_start = path.LastIndexOf('\\');
            string name = path.Substring(name_start+1);
            int index = Program.CurrentDirectory.SearchDirectory(name);
            if(index == -1)
            {
                if(size > 0 )
                {
                    FileEntry file = new FileEntry(name.ToCharArray(), 0x0, FAT_Table.Available_Block_In_FatTable(), size,Program.CurrentDirectory, content);
                    file.WriteFile();
                    DirectoryEntry dir = new DirectoryEntry(name.ToCharArray(), 0x0, FAT_Table.Available_Block_In_FatTable(), size);
                    Program.CurrentDirectory.DirectoryTable.Add(dir);
                    Program.CurrentDirectory.WriteDirctory();
                }
                else
                {
                    FileEntry file = new FileEntry(name.ToCharArray(), 0x0, 0,0 ,Program.CurrentDirectory, content);
                    file.WriteFile();
                }
            }
            else
            {
                Console.WriteLine("The File Already Exist");
            }
        }
        else
        {
            Console.WriteLine("This File Not Exist");
        }
    }
```

- **Type Class:**

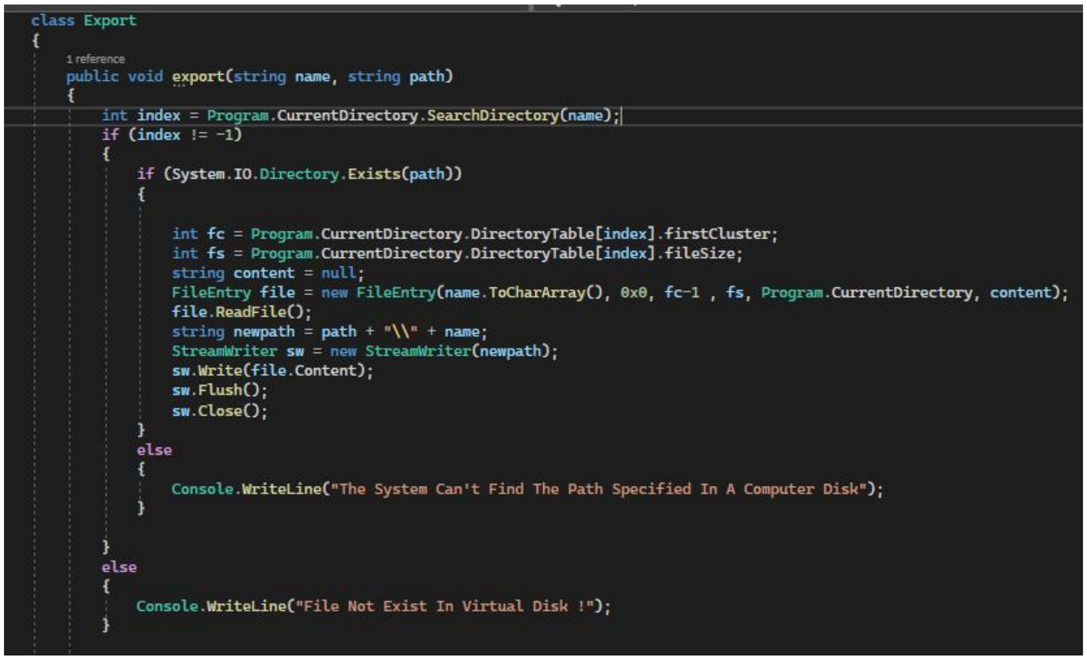
It displays the content of the file by check the existence of the file and display the content in the screen.



```
2 references
class Type
{
    1 reference
    public void type(string name)
    {
        int index = Program.CurrentDirectory.SearchDirectory(name);

        if ( index != -1)
        {
            int fc =  Program.CurrentDirectory.DirectoryTable[index].firstCluster;
            int fs = Program.CurrentDirectory.DirectoryTable[index].fileSize;
            string content = null;
            FileEntry file = new FileEntry(name.ToCharArray(),0x0,fc-1,fs,Program.CurrentDirectory,content);
            file.ReadFile();
            Console.WriteLine(file.Content);

        }
        else
        {
            Console.WriteLine("System Can't Find File Specify !");
        }
    }
}
```

- **Export Class:**

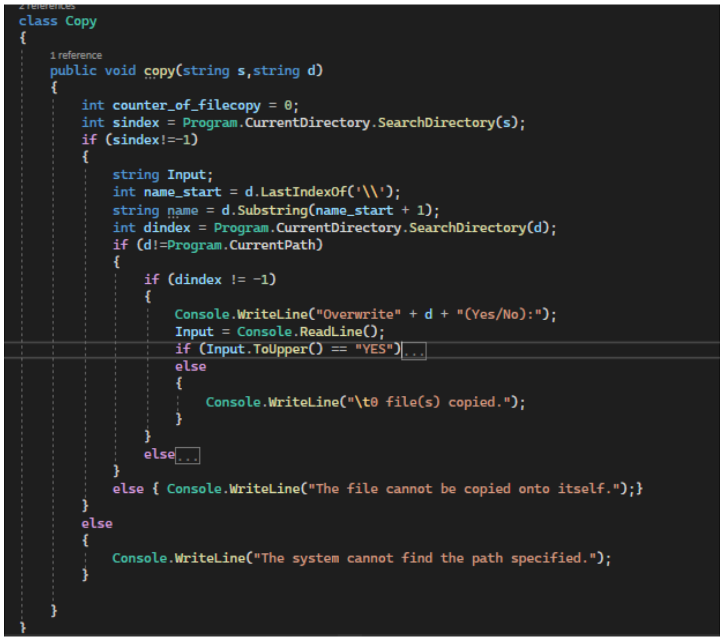
It displays the content of any file in virtual disk in a specified file in a valid path in computer.



```
class Export
{
    1 reference
    public void export(string name, string path)
    {
        int index = Program.CurrentDirectory.SearchDirectory(name);
        if (index != -1)
        {
            if (System.IO.Directory.Exists(path))
            {

                int fc = Program.CurrentDirectory.DirectoryTable[index].firstCluster;
                int fs = Program.CurrentDirectory.DirectoryTable[index].fileSize;
                string content = null;
                FileEntry file = new FileEntry(name.ToCharArray(), 0x0, fc-1 , fs, Program.CurrentDirectory, content);
                file.ReadFile();
                string newpath = path + "\\" + name;
                StreamWriter sw = new StreamWriter(newpath);
                sw.Write(file.Content);
                sw.Flush();
                sw.Close();
            }
            else
            {
                Console.WriteLine("The System Can't Find The Path Specified In A Computer Disk");
            }

        }
        else
        {
            Console.WriteLine("File Not Exist In Virtual Disk !");
        }
```

- **Class Copy:**

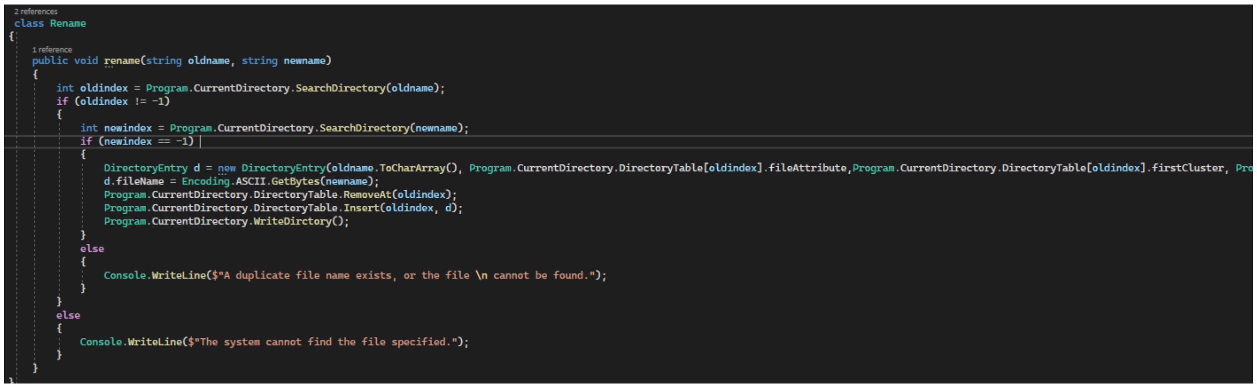
It responsible to copy the file from path to another valid path in virtual disk by checking the validity of that path.



```
2 references
class Copy
{
    1 reference
    public void copy(string s,string d)
    {
        int counter_of_filecopy = 0;
        int sindex = Program.CurrentDirectory.SearchDirectory(s);
        if (sindex!=-1)
        {
            string Input;
            int name_start = d.LastIndexOf('\\');
            string name = d.Substring(name_start + 1);
            int dindex = Program.CurrentDirectory.SearchDirectory(d);
            if (d!=Program.CurrentPath)
            {
                if (dindex != -1)
                {
                    Console.WriteLine("Overwrite" + d + "(Yes/No):");
                    Input = Console.ReadLine();
                    if (Input.ToUpper() == "YES")...
                    else
                    {
                        Console.WriteLine("\t0 file(s) copied.");
                    }
                }
                else...
            }
            else { Console.WriteLine("The file cannot be copied onto itself.");}
        }
        else
        {
            Console.WriteLine("The system cannot find the path specified.");
        }

    }
}
```

- **Rename Class:**

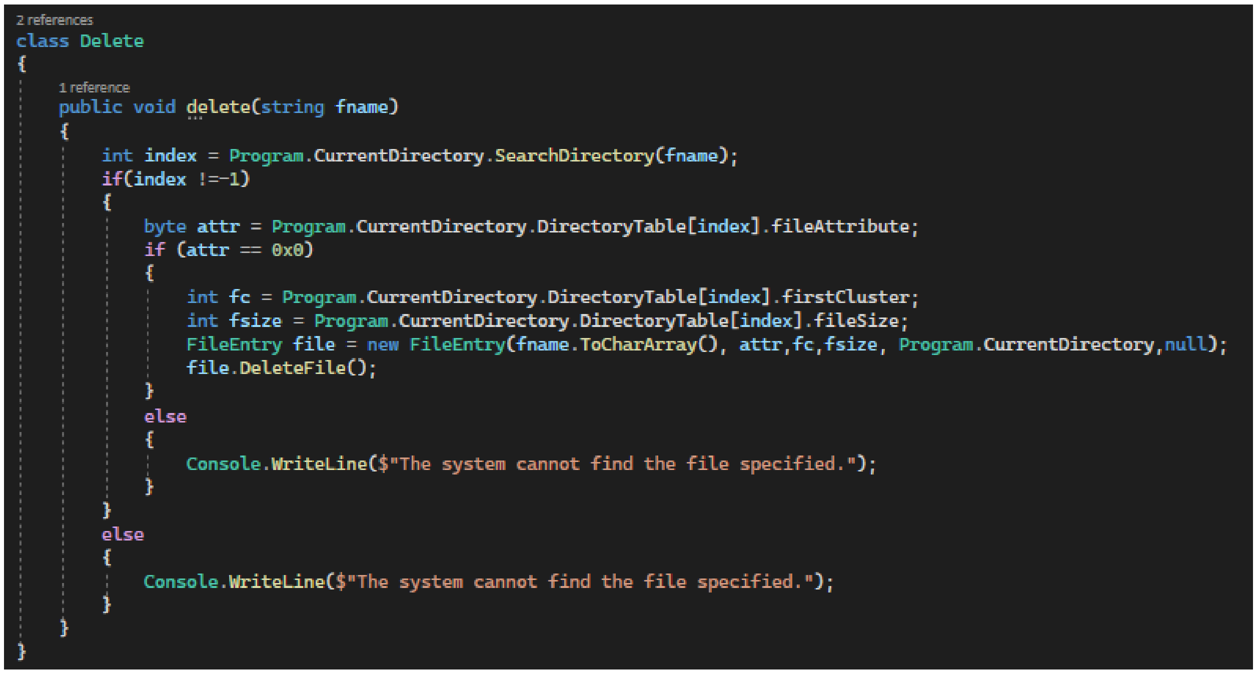
It renames the file by passing the old name and new name and remove the file with the old name from directory table and add the file with new name.



```
2 references
class Rename
{
    1 reference
    public void rename(string oldname, string newname)
    {
        int oldindex = Program.CurrentDirectory.SearchDirectory(oldname);
        if (oldindex != -1)
        {
            int newindex = Program.CurrentDirectory.SearchDirectory(newname);
            if (newindex == -1) |
            {
                DirectoryEntry d = new DirectoryEntry(oldname.ToCharArray(), Program.CurrentDirectory.DirectoryTable[oldindex].fileAttribute,Program.CurrentDirectory.DirectoryTable[oldindex].firstCluster, Pro
                d.fileName = Encoding.ASCII.GetBytes(newname);
                Program.CurrentDirectory.DirectoryTable.RemoveAt(oldindex);
                Program.CurrentDirectory.DirectoryTable.Insert(oldindex, d);
                Program.CurrentDirectory.WriteDirctory();
            }
            else
            {
                Console.WriteLine($"A duplicate file name exists, or the file \n cannot be found.");
            }
        }
        else
        {
            Console.WriteLine($"The system cannot find the file specified.");
        }
    }
}
```

- **Delete Class:**

This Class Delete a file from virtual disk by check the existence of that file and use function delete file in file entry



```
2 references
class Delete
{
    1 reference
    public void delete(string fname)
    {
        int index = Program.CurrentDirectory.SearchDirectory(fname);
        if(index !=-1)
        {
            byte attr = Program.CurrentDirectory.DirectoryTable[index].fileAttribute;
            if (attr == 0x0)
            {
                int fc = Program.CurrentDirectory.DirectoryTable[index].firstCluster;
                int fsize = Program.CurrentDirectory.DirectoryTable[index].fileSize;
                FileEntry file = new FileEntry(fname.ToCharArray(), attr,fc,fsize, Program.CurrentDirectory,null);
                file.DeleteFile();
            }
            else
            {
                Console.WriteLine($"The system cannot find the file specified.");
            }
        }
        else
        {
            Console.WriteLine($"The system cannot find the file specified.");
        }
    }
}
```

To see the code in details:

https://github.com/Randa-Adel/MiniFatSystem