

**GEBZE TECHNICAL UNIVERSITY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CSE 312 /CSE504**

**Operating Systems**

**Homework #2 Report**

**171044073**

**Berkan AKIN**

# **Requirement**

**1.1 Introduction and General Information**

* Assignment topic: Designing and implementing a simplified FAT-like file system.
* Programming language: C or C++.
* Maximum file size: 4 MB.
* File system type: FAT12-like structure.

## 1.2 Part 1: File System Design

* The file system will use the FAT12 structure.
* Supported block sizes: 0.5 KB and 1 KB.
* File attributes:
  + File name (any length is possible).
  + Size.
  + Owner permissions (Read (R) and Write (W) only, no Execute (X)).
  + Last modification date and time.
  + File creation date and time.
  + Password protection.

**Design Report Content:**

* Define your directory table and directory entries.
* Define how and where you keep the free blocks.
* Define how you solve arbitrary length of file names.
* Define how you handle the permissions.
* Define how you handle password protection.
* Include the function names of your source code that handle the file system operations listed in Part 3.

## 1.3 Part 2: Creating an Empty File System

* Write a C/C++ program to create an empty file system.
* Sample command: makeFileSystem 1 mySystem.dat
  + 1: Block size of the file system in KB.
  + mySystem.dat: Linux file containing all the file system information.
* File system information includes: super block, data blocks, free blocks, directories, etc.

## 1.4. Part 3: File System Operations

* Program to be written: fileSystemOper
* File system operations and parameters:
  + dir <path>: Lists the contents of the specified directory.
  + mkdir <path>: Creates a new directory.
  + rmdir <path>: Removes a directory.
  + dumpe2fs: Provides information about the file system.
  + write <path> <file>: Creates and writes data to a file in the file system.
  + read <path> <file>: Reads data from a file in the file system.
  + del <path> <file>: Deletes a file from the file system.
  + chmod <path> <file> <permissions>: Changes the owner permissions of a file.
  + addpw <path> <file> <password>: Adds a protection password to a file.

## 1.5 Test Commands

* Use the provided sample command sequence to test your file system operations:

shell

Kodu kopyala

makeFileSystem 4 mySystem.data

fileSystemOper fileSystem.data mkdir “\usr”

fileSystemOper fileSystem.data mkdir “\usr\ysa”

fileSystemOper fileSystem.data mkdir “\bin\ysa” ; Should print error!

fileSystemOper fileSystem.data write “\usr\ysa\file1” linuxFile.data

fileSystemOper fileSystem.data write “\usr\file2” linuxFile.data

fileSystemOper fileSystem.data write “\file3” linuxFile.data

fileSystemOper fileSystem.data dir “\” ; Should list 1 dir, 1 file

fileSystemOper fileSystem.data del “\usr\ysa\file1”

fileSystemOper fileSystem.data dumpe2fs

fileSystemOper fileSystem.data read “\usr\file2” linuxFile2.data

cmp linuxFile2.data linuxFile.data ; Should not print any difference

fileSystemOper fileSystem.data chmod “\usr\file2” -rw

fileSystemOper fileSystem.data read “\usr\file2” linuxFile2.data ;Should produce an error.

fileSystemOper fileSystem.data chmod “\usr\file2” +rw

fileSystemOper fileSystem.data addpw “\ysa\file2” test1234

fileSystemOper fileSystem.data read “\usr\file2” linuxFile2.data ;Should produce an error.

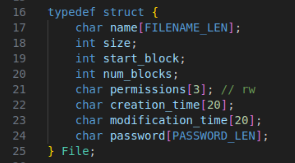
fileSystemOper fileSystem.data read “\usr\file2” linuxFile2.data test1234 ;Should be OK

In your report, you should explain these sections in detail and use the provided examples to test your file system. Prepare a report that details your file system design and implementation according to these requirements.

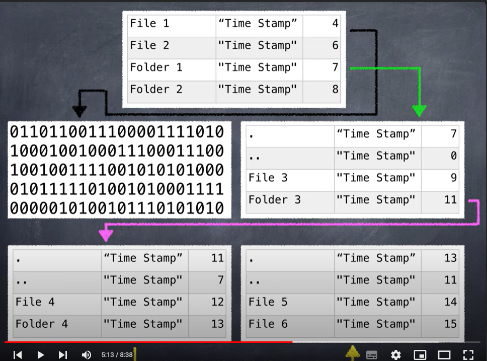
# **2.0 File System Design**

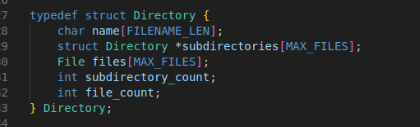
The file system design is based on the simplified FAT12 structure, as described in the given code. The design includes the following features:

1. **File Attributes**:
   * **File Name**: The file name can be of any length, but for simplicity, it is limited to 20 characters (FILENAME\_LEN).
   * **Size**: The size of the file is stored as an integer.
   * **Start Block**: The starting block of the file in the file system.
   * **Number of Blocks**: The number of blocks occupied by the file.
   * **Permissions**: Read (R) and Write (W) permissions are managed as a string of length 3.
   * **Creation and Modification Time**: Both are stored as strings of length 20.
   * **Password**: The password protection for the file is stored as a string of length 20.

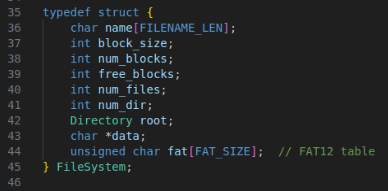


1. **Directory Structure**:
   * The directory structure supports nested directories and files within directories.
   * Each directory can have a maximum of 100 subdirectories and files (MAX\_FILES).
   * Directory information includes:
     + Name
     + Subdirectories (array of pointers to Directory)
     + Files (array of File)
     + Count of subdirectories and files.



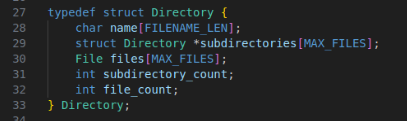


1. **File Allocation Table (FAT12)**:
   * The FAT12 table is implemented as an array of unsigned chars.
   * Each entry in the FAT12 table is 12 bits, and the table size is calculated as FAT\_SIZE = (MAX\_BLOCKS \* 3 / 2).
2. **File System Metadata**:
   * The file system metadata includes:
     + Name
     + Block size
     + Number of blocks
     + Free blocks
     + Number of files and directories
     + Root directory
     + Data block for storing file data
     + FAT12 table.

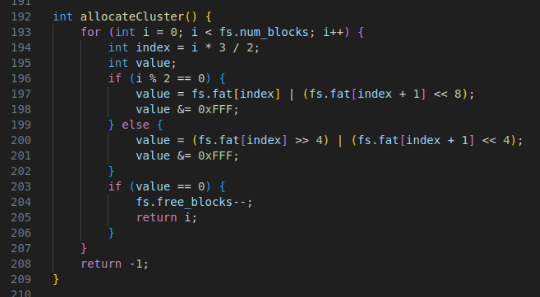


**Design Report Content**

**1. Directory Table and Directory Entries:** The directory table is defined within the Directory struct, which includes arrays for subdirectories and files. Each directory has a name, count of subdirectories, and count of files.



**2. Free Blocks Management:** Free blocks are managed using the FAT12 table. Each entry in the FAT table indicates whether a block is free or occupied by a file. The allocateCluster function finds the first free block and marks it as occupied.

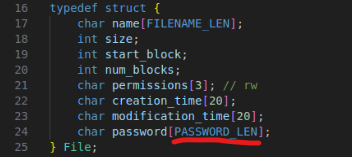


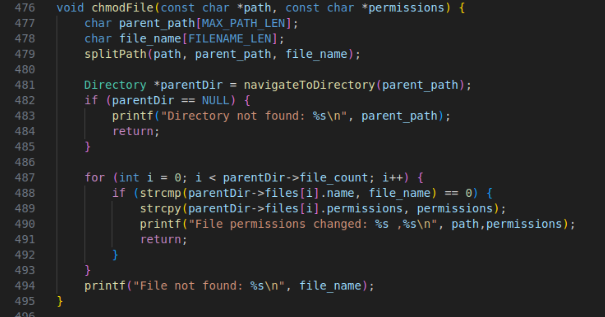
1. **Handling Arbitrary Length File Names:** The file name length is restricted to FILENAME\_LEN (20 characters) for simplicity. This limitation ensures that file names are easy to manage within the directory structure.



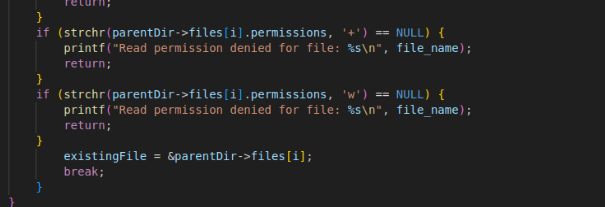


**4. Permissions Management:** Permissions are managed as a string of length 3, containing characters for read (r) and write (w) permissions. The chmodFile function updates the permissions for a specified file.

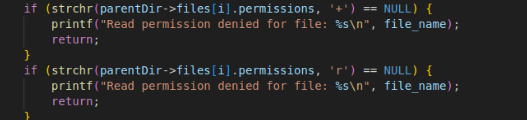




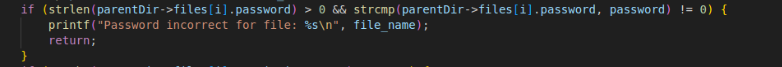
**Write permission**

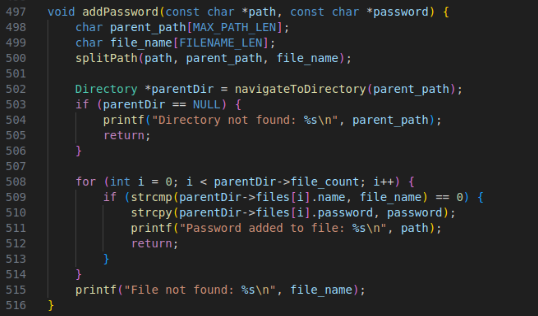


**Read Permission**



1. **Password Protection Management:** Password protection for files is handled by storing the password as a string of length PASSWORD\_LEN (20 characters). The addPassword function adds or updates the password for a specified file.





**6. File System Operations Functions:** The report should include the function names handling various file system operations:

* **Creating File System**: createFileSystem
* **Saving File System**: saveFileSystem
* **Loading File System**: loadFileSystem
* **Writing to File**: writeToFile
* **Reading from File**: readFromFile
* **Deleting File**: deleteFile
* **Listing Directory Contents**: listDirectory
* **Changing File Permissions**: chmodFile
* **Adding Password to File**: addPassword
* **Creating Directory**: makeDirectory
* **Removing Directory**: removeDirectory
* **Dumping File System Information**: dumpFileSystem



The design of this file system aligns with the requirements of a simplified FAT-like structure. By adhering to the constraints and functionalities specified, the file system ensures efficient management of files and directories, supports password protection, and handles read/write permissions effectively.

**Part 2: Creating an Empty File System**

This section outlines the steps and code necessary to create an empty file system. The empty file system is created as a file with the specified size and includes the file system's metadata, FAT12 table, and data blocks.

**1. Creating the File System**

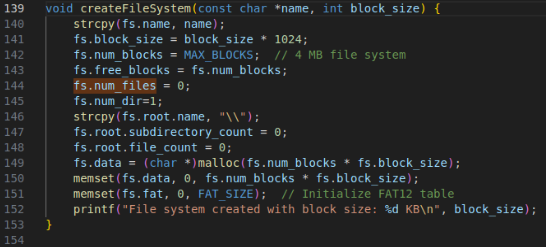
To create an empty file system, the createFileSystem function is used. This function initializes the necessary settings by taking the name and block size of the file system.

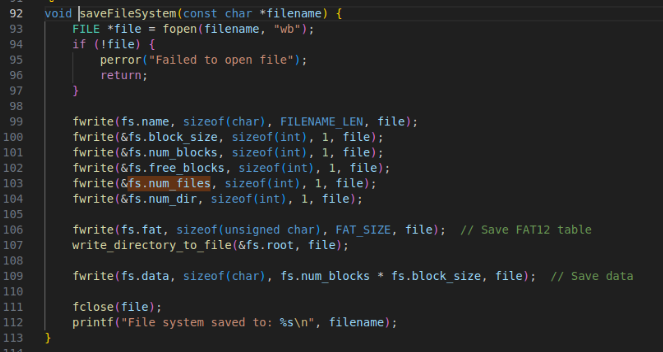
* fs.name: Stores the name of the file system.
* fs.block\_size: Stores the block size (in KB).
* fs.num\_blocks: Sets the maximum number of blocks (4096 blocks).
* fs.free\_blocks: Initially indicates that all blocks are free.
* fs.num\_files: Initially sets the number of files to zero.
* fs.num\_dir: Indicates that there is one directory initially (the root directory).
* fs.root: Stores the root directory's information.
* fs.data: Allocates memory for storing file data.
* fs.fat: Initializes the FAT12 table.



**2. Saving the File System**

The created file system is saved to a specified file. The saveFileSystem function writes the file system's metadata, FAT12 table, directory structures, and data blocks to the file.



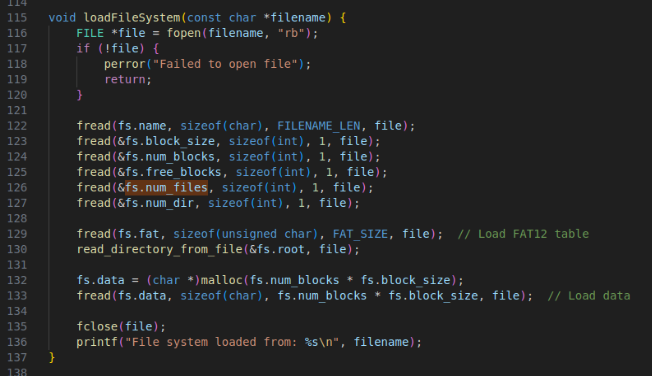


fwrite: Writes the file system's name, block size, number of blocks, number of free blocks, and number of files and directories to the file.

* write\_directory\_to\_file: Writes the root directory and its subdirectories' information to the file.
* fwrite: Writes the data blocks to the file.

**3. Loading the File System**

The saved file system is loaded from a specified file. The loadFileSystem function reads the file system's metadata, FAT12 table, directory structures, and data blocks from the file.



**4. Example Usage**

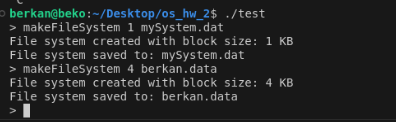
To create and save an empty file system, the following command is used:

shell

Kodu kopyala

makeFileSystem 4 mySystem.dat

* 4: Specifies the block size (in KB).
* mySystem.dat: The name of the file to store the file system.







This command creates a file system with a 4 KB block size and saves it to a file named mySystem.dat.

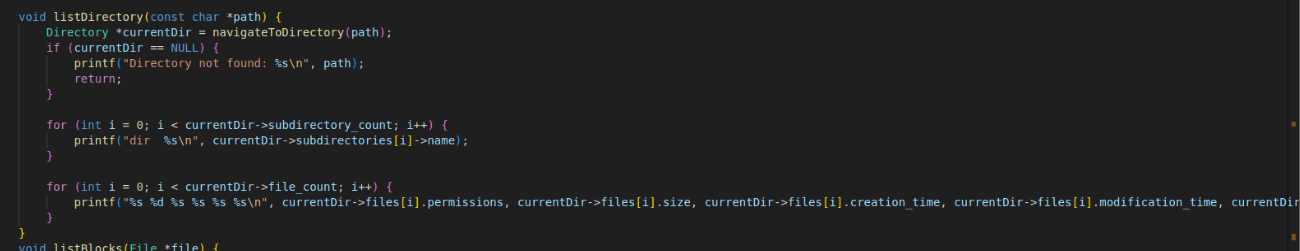
The file system is created and saved using the createFileSystem and saveFileSystem functions. This process writes the file system's metadata, FAT12 table, and data blocks to the specified file, making the file system fully functional.

**Part 3: File System Operations**

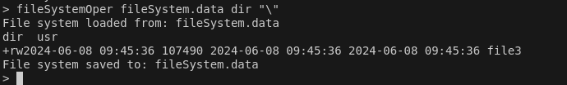
In this section, the functions and operations necessary for managing the file system are outlined. The file system operations include creating directories, listing directory contents, writing to and reading from files, deleting files, changing file permissions, and adding passwords to files.

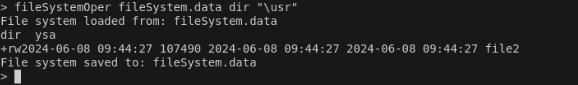
**1. Listing Directory Contents**

The listDirectory function lists the contents of a specified directory, including subdirectories and files with their attributes.



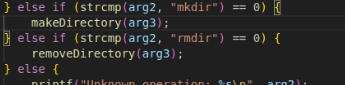
**Test**



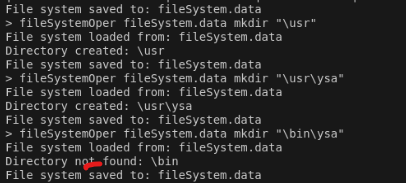


**2. Creating and Removing Directories**

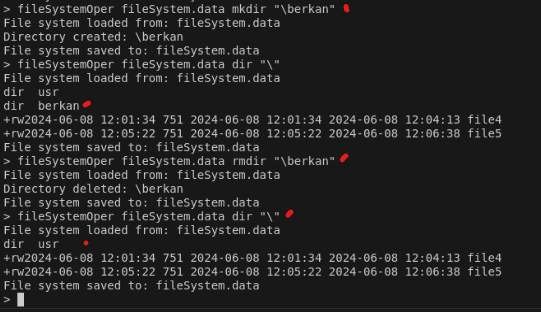
The makeDirectory function creates a new directory, while the removeDirectory function removes an existing directory.



**Mkdir Test**

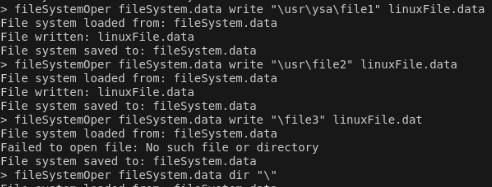


**Rmdir Test**

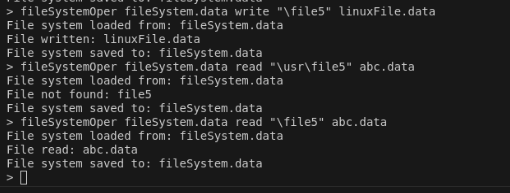
****

**3. Writing to and Reading from Files**

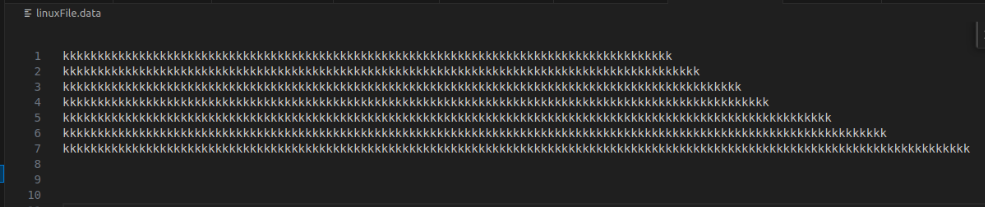
The writeToFile function writes data to a file in the file system, while the readFromFile function reads data from a file in the file system.



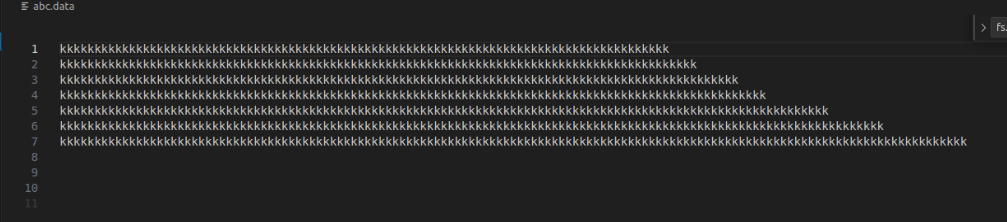
**Writing Test and Reading Test**

****

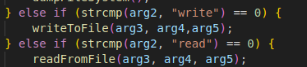
**linucFile.data**

****

**abc.data**

****

****

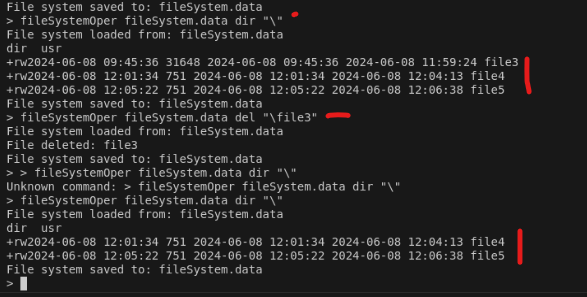
****

**4. Deleting Files**

The deleteFile function deletes a file from the file system by freeing its blocks and removing it from the directory structure.

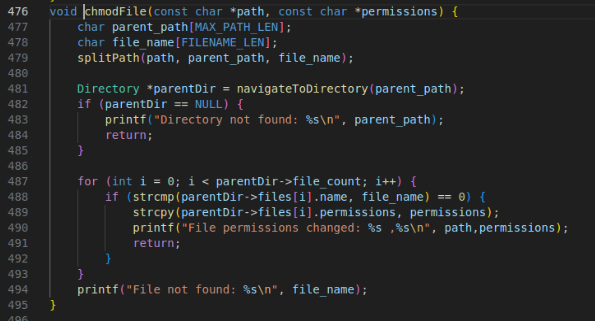


**file3 deleted**

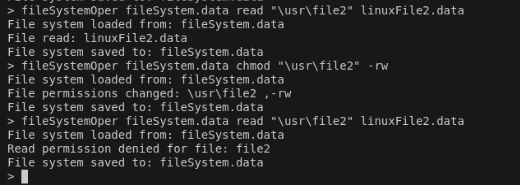


**5. Changing File Permissions**

The chmodFile function changes the permissions of a specified file.

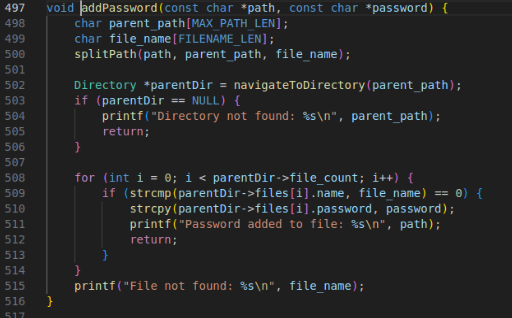


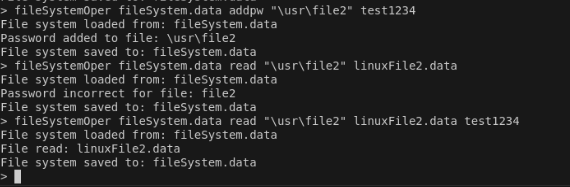
file2 was first opened with read and write privileges, but could not be opened after the authorization was obtained.

****

**6. Adding Passwords to Files**

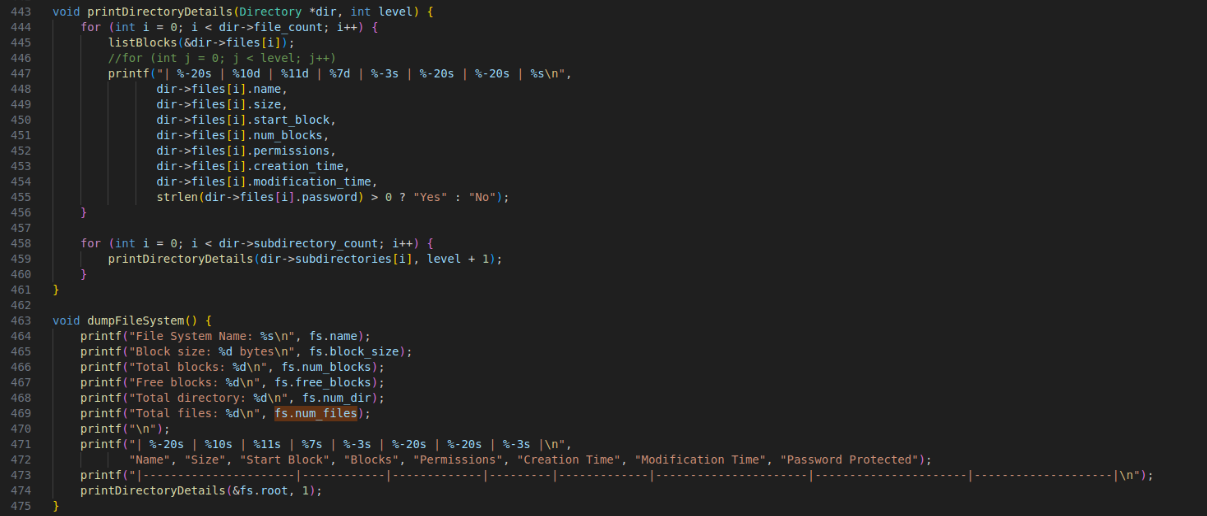
The addPassword function adds or updates a password for a specified file.

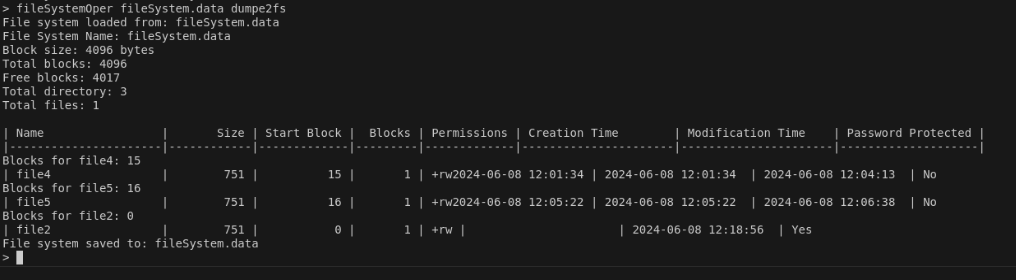




**7. Dumping File System Information**

The dumpFileSystem function provides detailed information about the file system, including block usage and file attributes.





The file system operations outlined in this section provide comprehensive functionality for managing files and directories in the FAT12-like file system. These operations include listing directory contents, creating and removing directories, writing to and reading from files, deleting files, changing file permissions, and adding passwords to files. Each function is designed to interact with the file system's metadata, FAT12 table, and data blocks to ensure efficient and secure file management.