**Ex. No.: 18**

**Date:**

**FILE ORGANISATION TECHNIQUES**

**a. SINGLE LEVEL:**

**CODE:**

dir = {

    'dname': '',

    'files':{}

}

dir['fcnt'] = 0

dir['dname'] = input("Enter name of directory -- ")

while True:

    print("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Display file content\t6. Exit")

    ch = int(input("Enter your choice -- "))

    if ch == 1:

        fname = input("\n Enter the name of the file -- ")

        content=input("Enter file contents --")

        dir['files'][fname]=content

    elif ch == 2:

        f = input("\n Enter the name of the file -- ")

        if f in dir['files']:

            del dir['files'][f]

            print("File",f,"found and deleted")

        else:

            print("File", f, "not found")

            dir['fcnt'] -= 1

    elif ch == 3:

        f = input("\n Enter the name of the file -- ")

        if f in dir['files']:

            print("File",f,"found")

        else:

            print("File", f, "not found")

    elif ch == 4:

        if len(dir['files'])==0:

            print("\n Directory Empty")

        else:

            print("\n The Files are -- ")

            for i in dir['files']:

                print(i,end="\n")

    elif ch == 5:

        f = input("\n Enter the name of the file -- ")

        if f in dir['files']:

            print("Content:")

            print(dir['files'][f])

        else:

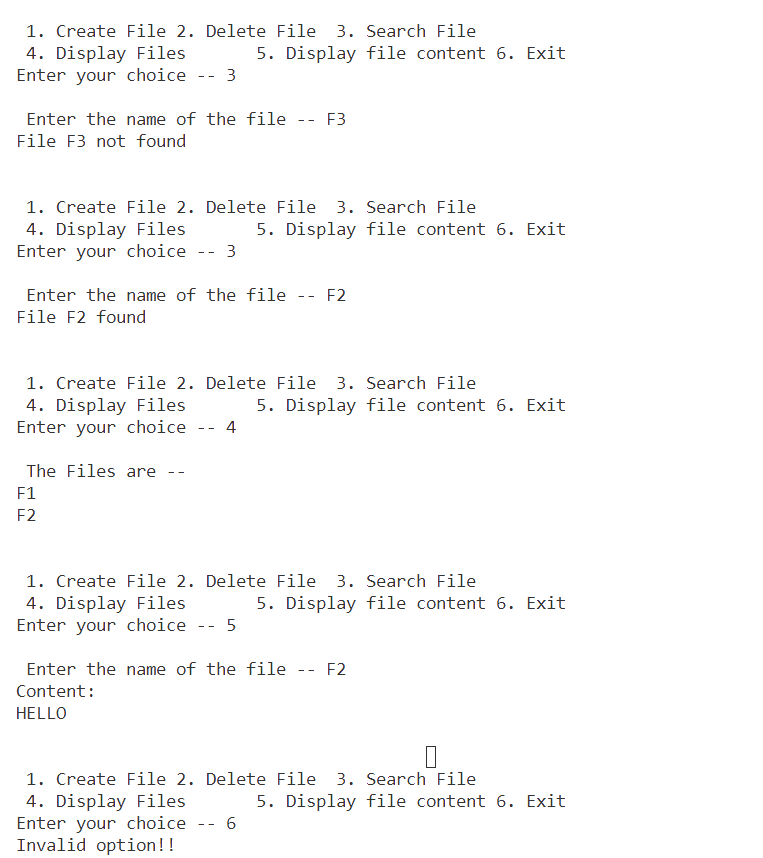
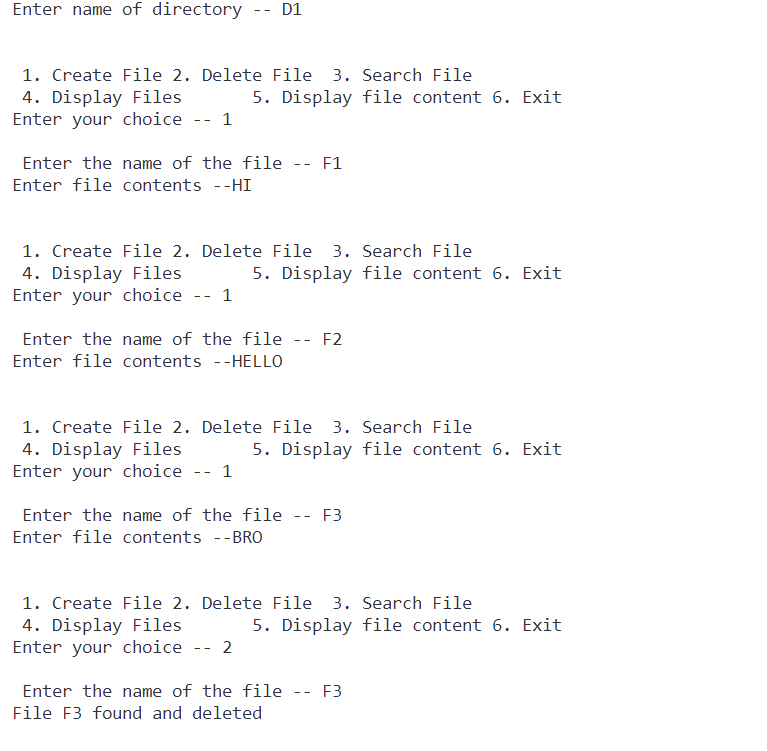
            print("File not found")

    else:

        print("Invalid option!!")

        break

**OUTPUT:**

****

**c. TREE LEVEL:**

**CODE:**

class Node:

    def \_\_init\_\_(self, name, type):

        self.name = name

        self.type = type

        self.next = None

        self.down = None

        self.content = None

def new\_node(item, type1, content=None):

    temp = Node(item, type1)

    temp.next = None

    temp.down = None

    temp.content = content

    return temp

def inorder(root, p):

    if root.next != None and root.name != p:

        inorder(root.next, p)

        print(root.name)

        if root.type == 1:

            inorder(root.down, p)

    return root

def find(node, key):

    if node is not None:

        if node.name == key:

            print(f"Found {node.name}")

            return node

        else:

            found\_node = find(node.down, key)

            if found\_node is None:

                found\_node = find(node.next, key)

            return found\_node

    return None

def insert(node, key, par, mode, content=None):

    if node is None:

        print("The root node is getting created.....")

        return new\_node(key, mode, content)

    else:

        temp = None

        temp = inorder(node, par)

        temp1 = new\_node(key, mode, content)  # Pass content parameter here

        if temp.down is None and temp.type == 1:

            temp.down = temp1

            if temp1.type == 2:

                print(f"File {temp1.name} successfully inserted")

            else:

                print(f"Directory {temp1.name} successfully inserted")

        else:

            temp = temp.down

            while temp.next is not None:

                temp = temp.next

            temp.next = temp1

            if temp1.type == 2:

                print(f"File {temp1.name} successfully inserted")

            else:

                print(f"Directory {temp1.name} successfully inserted")

    return node

root = None

c = 0

p = 0

parent = [None] \* 50

child = [None] \* 50

cont = 'y'

root = insert(root, "root", "", 1)

while cont == 'y':

    par\_dir = input("Enter parent directory: ")

    t = int(input("Enter type (1 for directory and 2 for file): "))

    file\_or\_dir = input("Enter directory or file name: ")

    # Ask for file contents if it's a file

    content = None

    if t == 2:

        content = input("Enter file contents: ")

    insert(root, file\_or\_dir, par\_dir, t, content)

    child[c] = file\_or\_dir

    parent[p] = par\_dir

    c += 1

    p += 1

    cont = input("Wanna insert more? (y/n): ")

finder = input("Enter file name/directory name to search: ")

found\_node = find(root, finder)

if found\_node:

    option = input("Do you want to display file content? (y/n): ")

    if found\_node.content==None:

        print("It's a directory")

    elif option.lower() == 'y' and found\_node.type == 2 and found\_node.content:

        print(f"File content of {found\_node.name}: {found\_node.content}")

        par = ""

        chi = found\_node.name

        print("The path in reverse order is")

        while par != "root":

            for i in range(c):

                if child[i] == chi:

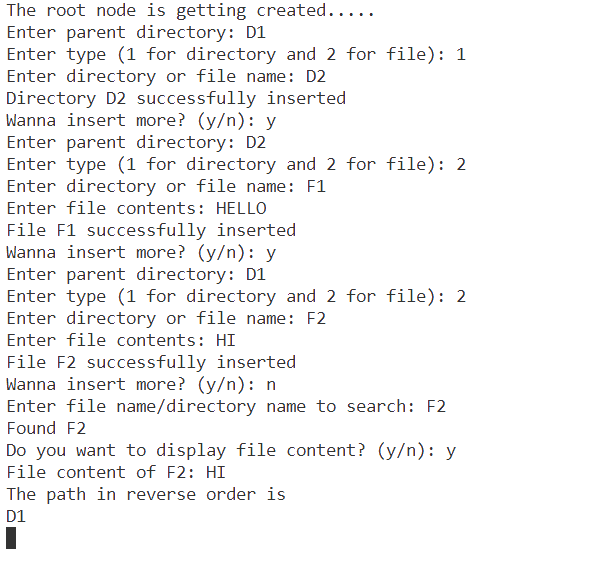
                    par = parent[i]

                    chi = parent[i]

                    print(par)

                    break

**OUTPUT:**

****

**B. TWO LEVEL**

import os

class File:

    def \_\_init\_\_(self, filename, content):

        self.filename = filename

        self.content = content

class UserDirectory:

    def \_\_init\_\_(self, path):

        self.path = path

        self.files = {}

def create\_user\_directory(system, master\_directory, username):

 # Create a new user directory for the given username.

    if username not in system.master\_directory:

        user\_path = os.path.join(master\_directory, username)

        os.makedirs(user\_path, exist\_ok=True)

        system.master\_directory[username] = UserDirectory(user\_path)

def create\_new\_file\_in\_user\_directory(system, username, filename, content):

 # Create a new file with content in the user's directory.

    if username in system.master\_directory:

        user\_directory = system.master\_directory[username]

        file\_path = os.path.join(user\_directory.path, filename)

        with open(file\_path, 'w') as file:

            file.write(content)

        user\_directory.files[filename] = File(filename, content)

    else:

        print(f"User '{username}' not found. Create the user directory first.")

def list\_user\_files(system, username):

 # List the files in the user's directory.

    if username in system.master\_directory:

        user\_directory = system.master\_directory[username]

        print(f"Files in User '{username}' Directory:")

        for filename in user\_directory.files:

            print(filename)

def read\_file(system, username, filename):

 # Read the content of a file in the user's directory.

    if username in system.master\_directory:

        user\_directory = system.master\_directory[username]

        if filename in user\_directory.files:

            file = user\_directory.files[filename]

            return file.content

        else:

            return f"File '{filename}' not found in User '{username}' Directory."

    else:

        return f"User '{username}' not found."

# Interactive driver code

class TwoLevelDirectorySystem:

    def \_\_init\_\_(self):

 # The master directory, a dictionary with usernames as keys and user directories as values.

        self.master\_directory = {}

master\_directory\_path = input("Enter the path where the master directory should be created: ")

if not os.path.exists(master\_directory\_path):

     os.makedirs(master\_directory\_path)

system = TwoLevelDirectorySystem()

while True:

    print("\nOptions:")

    print("1. Create User Directory")

    print("2. Create New File in User Directory")

    print("3. List User Files")

    print("4. Read File Content")

    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':

        username = input("Enter the username: ")

        create\_user\_directory(system, master\_directory\_path, username)

    elif choice == '2':

        username = input("Enter the username: ")

        filename = input("Enter the filename: ")

        content = input("Enter the content for the new file: ")

        create\_new\_file\_in\_user\_directory(system, username, filename, content)

    elif choice == '3':

        username = input("Enter the username: ")

        list\_user\_files(system, username)

    elif choice == '4':

        username = input("Enter the username: ")

        filename = input("Enter the filename: ")

        content = read\_file(system, username, filename)

        print(content)

    elif choice == '5':

        break

    else:

        print("Invalid choice. Please enter a valid option.")

**D. ACYCLIC GRAPH**

class File:

    def \_\_init\_\_(self, path):

        self.path = path

class Directory:

    def \_\_init\_\_(self, dname):

        self.dname = dname

        self.files = []

def create\_directory(path, dname, files):

    directory\_path = os.path.join(path, dname)

    os.makedirs(directory\_path, exist\_ok=True)

    directory = Directory(directory\_path)

    for file in files:

        directory.files.append(file)

    return directory

def search\_file(fname):

    matches = []

    for directory in directories:

        for file in directory.files:

            if fname in file.path:

                matches.append((directory.dname, file.path))

    if matches:

        print("\nMatch(es) found:")

        for directory\_name, match in matches:

            print(f"In Directory '{directory\_name}': {match}")

count = int(input("Enter the number of base directories: "))

directories = []

for \_ in range(count):

    base\_path = input("Enter the base directory path: ")

    dname = input("Enter the directory name: ")

    fcount = int(input("Enter the number of files in the directory: "))

    files = []

    for \_ in range(fcount):

        path = input("Enter file path: ")

        files.append(File(path))

    directories.append(create\_directory(base\_path, dname, files))

search\_key = input("Enter the file to search: ")

search\_file(search\_key)