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:

```
Enter name of directory -- D1


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 1

 Enter the name of the file -- F1
Enter file contents --HI


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 1

 Enter the name of the file -- F2
Enter file contents --HELLO


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 1

 Enter the name of the file -- F3
Enter file contents --BRO


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 2

 Enter the name of the file -- F3
File F3 found and deleted
```

```
 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 3

 Enter the name of the file -- F3
File F3 not found


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 3

 Enter the name of the file -- F2
File F2 found


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 4

 The Files are --
F1
F2


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 5

 Enter the name of the file -- F2
Content:
HELLO


 1. Create File 2. Delete File  3. Search File
 4. Display Files       5. Display file content 6. Exit
Enter your choice -- 6
Invalid option!!
```

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:

```
The root node is getting created.....
Enter parent directory: D1
Enter type (1 for directory and 2 for file): 1
Enter directory or file name: D2
Directory D2 successfully inserted
Wanna insert more? (y/n): y
Enter parent directory: D2
Enter type (1 for directory and 2 for file): 2
Enter directory or file name: F1
Enter file contents: HELLO
File F1 successfully inserted
Wanna insert more? (y/n): y
Enter parent directory: D1
Enter type (1 for directory and 2 for file): 2
Enter directory or file name: F2
Enter file contents: HI
File F2 successfully inserted
Wanna insert more? (y/n): n
Enter file name/directory name to search: F2
Found F2
Do you want to display file content? (y/n): y
File content of F2: HI
The path in reverse order is
D1
```

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)
```

Ex. No: 19

Date:

INDEXED FILE ALLOCATION

Problem Statement:

To write a python program to implement Indexed file allocation method

Problem Description:

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.

Algorithm:

Step 1: Initialize the disk and data structures

Read the number of blocks in the disk from the user and store it in 'num_block'.

- Create a list 'disk' of length 'num_block' and initialize it with -1 to represent free blocks.
- Create an array 'arr' to store file block mappings.
- Create a list 'file_map' to store file names.
- Create a list 'map_idx' to store the index block for each file.
- Create a list 'sz' to store the total number of blocks in each file.
- Initialize 'tot_file' to 0 to keep track of the total number of files.

Step 2: Start an infinite loop to display the menu and handle user choices.

Step 3: Print the menu options: To add a file, to print the directory, to exit

Step 4: Handle user choice

- Read the user's choice (integer) and store it in the 'choice' variable.
- Use conditional statements to handle different choices:

If 'choice' is 1:

- Read the file name, indexed block, and the total number of blocks in the file from the user.
- Validate the indexed block to ensure it is not occupied by another file.
- Find and allocate free blocks for the file's data blocks.
- Update data structures to store the file's information (name, index block, and size).
- Increment 'tot_file' to reflect the addition of a new file.

If 'choice' is 2:

- Print the directory showing file names, index blocks, and the blocks stored for each file.

If 'choice' is 3:

- Exit the program.

If 'choice' is not in the range 1-3, print "Invalid Input."

Step 5: Exit the program

Code:

```
def print_menu():
    print("Enter:")
    print("1. To add File")
    print("2. To Print Directory")
    print("3. To exit")
if __name__== "__main__":
    num_block = int(input("Enter the number of blocks in the disk: "))
    disk = [-1] * num_block
    arr = [[None] * 100000 for _ in range(1000)]
    file_map = [None] * 100000
    map_idx = [0] * 100000
    sz = [0] * 100000
    tot_file = 0
    print("Welcome to the indexed File")
    while True:
        print_menu()
        choice = int(input())
        if choice == 1:
            file_name, i_block, t_block = input("Enter File name, indexed block, and total number of blocks in the file: ").split()
            i_block, t_block = int(i_block), int(t_block)
            if i_block < 0 or i_block >= num_block or disk[i_block] != -1:
```

```
            print("Index Block is not empty or invalid")
            continue
        free_block = []
        j = 0
        cnt = 0
        disk[i_block] = tot_file
        for i in range(t_block):
            while j < num_block:
                if disk[j] == -1:
                    free_block.append(j)
                    cnt += 1
                    j += 1
                    break
                j += 1
        if cnt == t_block:
            for i in range(t_block):
                arr[tot_file][i] = free_block[i]  # Store free block index
                disk[free_block[i]] = tot_file
            print()
            file_map[tot_file] = file_name
            map_idx[tot_file] = i_block
            sz[tot_file] = t_block
            tot_file += 1
        else:
            disk[i_block] = -1
            print("Not Enough free Space")
    elif choice == 2:
        print("File name        Index Block     Block Stored")
        for i in range(tot_file):
            print(f"{file_map[i]:<16} {map_idx[i]:<16}", end="")
```

```
            for j in range(sz[i]):
                print(f"{arr[i][j]}", end="")
                if j < sz[i] - 1:
                    print(" --> ", end="")
            print()
        elif choice == 3:
            break
        else:
            print("Invalid Input")
```

Output:

```
Enter the number of blocks in the disk: 30
Welcome to the indexed File
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file1.txt 15 6

Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name       Index Block     Block Stored
file1.txt       15              0 --> 1 --> 2 --> 3 --> 4 --> 5
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file2 29 23
Not Enough free Space
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file2.txt 29 22
```

```
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name       Index Block     Block Stored
file1.txt       15              0 --> 1 --> 2 --> 3 --> 4 --> 5
file2.txt       29              6 --> 7 --> 8 --> 9 --> 10 --> 11 --> 12 --> 13 --> 14 --> 16 --> 17 --> 18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26 --> 27
 --> 28
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file3.txt 22 5
Index Block is not empty or invalid
Enter:
1. To add File
2. To Print Directory
3. To exit
3
```

Ex. No: 19

Date:

FILE ALLOCATION TECHNIQUES USING LINKED LIST

Problem Statement:

To implement file allocation using linked list.

Problem Description:

Linked File Allocation is a Non-contiguous memory allocation method where the file is stored in random memory blocks and each block contains the pointer (or address) of the next memory block as in a linked list. The starting memory block of each file is maintained in a directory and the rest of the file can be traced from that starting block.

Algorithm:

1. Initialize data structures to represent disk blocks, file names, and file metadata.
2. Display a menu for the user with options to add a file, print the directory, or exit.
3. In an infinite loop, wait for the user to choose an option.
4. If the user chooses to add a file:
    - Prompt for the file name, starting block, and total blocks.
    - Check if the file name exists, the starting block is valid and available, and if there are enough free blocks.
    - Allocate blocks to the file, update data structures, and keep track of the total files.
5. If the user chooses to print the directory:
    - Display the list of file names and their allocated blocks.
6. If the user chooses to exit, terminate the program.
7. If the user provides an invalid option, display an error message.
8. End the loop when the user exits the program.

Code:

```
import random


class Node:
    def __init__(self, val): self.ptr
        = None
```

```
        self.val = val

arr = [None] * 100 fileMap =
[None] * 100 mapIdx = [0] *
100
sz = [0] * 100

def printMenu(): print("Enter:")
    print("1. To add File")
    print("2. To Print Directory") print("3. To
    exit")

def main():
    numBlock = int(input("Enter the number of Block in disk: ")) disk = [-1] *
    numBlock
    totFile = 0 file=[]
    blocks=[] count=0

    while True:
        printMenu()
        choice = int(input())

        if choice == 1:
            fName = input("Enter File name: ") if
            fName in file:
                print("File already exists") continue
            sBlock = int(input("Enter the starting block: "))
            if sBlock < 0 or sBlock >= numBlock or disk[sBlock] != -1:
```

```
            print("Start Block is not empty or invalid") continue
        tBlock = int(input("Enter the total number of blocks in the
file: "))        if count>numBlock:
            print("Enter valid no of blocks!") continue
        file.append(fName) freeBlock =
        [sBlock] blocks.append(sBlock)

        for _ in range(tBlock - 1): while
            True:
                randomBlock = random.randint(0, numBlock - 1)
                if disk[randomBlock] == -1 and randomBlock not in disk and
randomBlock not in blocks:
                    freeBlock.append(randomBlock)
                    disk[randomBlock] = totFile
                    count+=1
                    blocks.append(randomBlock) break

        head = None
        for block in freeBlock: idx =
            Node(block) idx.ptr =
            None
            if head is None: arr[totFile] =
                idx head = idx
            else:
                head.ptr = idx head =
                idx
```

```
                fileMap[totFile] = fName
                sz[totFile] = tBlock totFile += 1
                if count==numBlock: break

            elif choice == 2:
                print("File name               Block Stored")
                for i in range(totFile):
                    print(f"{fileMap[i]}                    ", end="")
                    head = arr[i]
                    while head is not None: print(f"{head.val} -
                        >", end="") head = head.ptr
                    print("NULL")

            elif choice == 3: break

            else:
                print("Invalid Input")

if __name____ == "__main__":
    main()
```

Output:

```
Enter the number of Block in disk: 30
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file1.txt
Enter the starting block: 4
Enter the total number of blocks in the file: 7
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name     Block Stored
file1.txt        4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file2.txt
Enter the starting block: 8
Enter the total number of blocks in the file: 7
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name     Block Stored
file1.txt        4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL
file2.txt       8 ->25 ->2 ->7 ->20 ->28 ->6 ->NULL
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file2.txt
File already exists
```

```
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file3.txt
Enter the starting block: 0
Enter the total number of blocks in the file: 6
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name     Block Stored
file1.txt        4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL
file2.txt       8 ->25 ->2 ->7 ->20 ->28 ->6 ->NULL
file3.txt       0 ->26 ->15 ->3 ->24 ->19 ->NULL
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file4.txt
Enter the starting block: 20
Start Block is not empty or invalid
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file4.txt
Enter the starting block: 1
Enter the total number of blocks in the file: 5
Enter:
1. To add File
2. To Print Directory
3. To exit
2
```

```
File name     Block Stored
file1.txt        4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL
file2.txt       8 ->25 ->2 ->7 ->20 ->28 ->6 ->NULL
file3.txt       0 ->26 ->15 ->3 ->24 ->19 ->NULL
file4.txt       1 ->12 ->22 ->21 ->17 ->NULL
```

Ex. No: 19

Date:

SEQUENTIAL/CONTIGUOUS FILE ALLOCATION METHOD

Problem Statement:

To write a python program to implement Sequential/Contiguous File Allocation Techniques.

Problem Description:

In this allocation strategy, each file occupies a set of contiguously blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file. The main problem with this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files.

Algorithm:

1.Initialize a list called disk with num_blocks elements, all initialized to -1, representing the storage blocks on the virtual disk.

2.Create dictionaries (file_map, map_idx, map_end_idx) to store file information, including file names, starting blocks, and ending blocks.
Initialize tot_file to 0, representing the total number of allocated files.

3.Start an infinite loop for user interaction.
Inside the loop, display a menu of options for the user to choose from.
(1: Add File, 2: Print Directory, 3: Exit)
Prompt the user to enter their choice by accepting an integer input.
Based on the user's choice, execute the corresponding functionality.

4.If the user chooses to add a file:
Enter the File name Starting block Total number of blocks required for the file.

5.Initialize a flag (flag) to 0.

6.Check if the requested blocks are available and if the file size can fit within the available space

7.Iterate through the disk from the start_block to check block availability. If a block is already allocated, break out of the loop.

8.If the loop completes without breaking and total_blocks are greater than 0, set flag to 1.

9.If flag is 1 (allocation is possible):

Allocate the blocks on the disk with a unique file identifier (an integer) for the file.

Store the file information in the dictionaries (file_map, map_idx, map_end_idx) and update tot_file.

10.Print a message indicating the allocated blocks and their range.

11.If flag is 0 (allocation is not possible):Display an error message.

12.Print Directory :Display the directory information in a well-formatted table. Include file names, starting blocks, ending blocks, and the blocks occupied by each file (presented as a range).

13.Exit Program :If the user chooses to exit the program, break out of the loop to terminate the program.

Code:

```
def print_menu():
    print("\nEnter:")
    print("1. To add File")
    print("2. To Print Directory")
    print("3. To exit")


if __name__ == "__main__":
    num_blocks = int(input("Enter the number of Blocks in the disk: "))
    disk = [-1] * num_blocks
    file_map = {}
    map_idx = {}
    map_end_idx = {}
    tot_file = 0


    print("Welcome to the Sequential File")


    while True:
        print_menu()
        choice = int(input("Enter your choice: "))
```

```
    if choice == 1:
        file_name = input("Enter File name: ")
        start_block, total_blocks = map(int, input("Enter starting block and total number of blocks in the file: ").split())
        flag = 0

        for i in range(start_block, num_blocks):
            if disk[i] != -1:
                break
            if start_block + total_blocks - 1 == i:
                flag = 1
                break

        if total_blocks == 0:
            flag = 0

        if flag == 1:
            occupied_blocks = [str(block) for block in range(start_block, start_block + total_blocks)]
            disk[start_block:start_block + total_blocks] = [tot_file] * total_blocks

            file_map[tot_file] = file_name
            map_idx[tot_file] = start_block
            map_end_idx[tot_file] = start_block + total_blocks - 1
            tot_file += 1
            print(f"File '{file_name}' allocated from block {start_block} to {start_block + total_blocks - 1}.")
        else:
            print("Something Went Wrong, either someone has occupied that space or out of bounds.")
    elif choice == 2:
        print("\n{:<15} {:<15} {:<15} {:<15}".format("File name", "Start Block", "End Block", "Blocks Occupied"))
```

```
        for i in range(tot_file):
            start_block = map_idx[i]
            end_block = map_end_idx[i]
            occupied_blocks = [str(block) for block in range(start_block, end_block + 1)]
            print("{:<15} {:<15} {:<15} {:<15}".format(file_map[i], start_block, end_block, ' -
-> '.join(occupied_blocks)))
    elif choice == 3:
        break
    else:
        print("Invalid Input")
```

Output:

```
Enter the number of Blocks in the disk: 30
Welcome to the Sequential File

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file1
Enter starting block and total number of blocks in the file: 18 9
File 'file1' allocated from block 18 to 26.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name       Start Block     End Block       Blocks Occupied
file1           18              26              18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file2
Enter starting block and total number of blocks in the file: 27 3
File 'file2' allocated from block 27 to 29.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name       Start Block     End Block       Blocks Occupied
file1           18              26              18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26
file2           27              29              27 --> 28 --> 29

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
```

```
Enter File name: file3
Enter starting block and total number of blocks in the file: 12 6
File 'file3' allocated from block 12 to 17.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name       Start Block     End Block       Blocks Occupied
file1           18              26              18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26
file2           27              29              27 --> 28 --> 29
file3           12              17              12 --> 13 --> 14 --> 15 --> 16 --> 17

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file4
Enter starting block and total number of blocks in the file: 2 10
File 'file4' allocated from block 2 to 11.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name       Start Block     End Block       Blocks Occupied
file1           18              26              18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26
file2           27              29              27 --> 28 --> 29
file3           12              17              12 --> 13 --> 14 --> 15 --> 16 --> 17
file4           2               11              2 --> 3 --> 4 --> 5 --> 6 --> 7 --> 8 --> 9 --> 10 --> 11

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file5
Enter starting block and total number of blocks in the file: 0 6
Something Went Wrong, either someone has occupied that space or out of bounds.
```