IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : N. Ravi Chandrika

Class: CSD-B Semester: Third Semester

Course Code : ACSD10 Course Name : OS Laboratory

Name of the Course Faculty: Ms. G. Indu Faculty ID : IARE10971

Exercise Number : 3 Week Number : 3 Date : 20-09-2024

Roll Number									
2	3	9	5	1	A	6	7	B	3

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab		Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	3.1	Managing student records in school database	4	2	2	4	3	4	19
2	3.2	managing medical records in hospital	4	2	2	4	4	4	20
3	3.3	Managing media files in multimedia	4	2	2	4	4	4	20
4	3.4	Digital Archive	4	2	2	4	4	4	20
5	3.5	Enterprise X	4	2	2	4	4	4	20
6									
7									
8									
9									
10									
11									
12									

N. Ravichandrika
Signature of the Student

Signature of the Faculty

3.1 Aim: Managing student records in a school database.

Code:

```
class StudentRecord:
    def __init__(self, name, student_id, grade, address):
        self.name = name
        self.student_id = student_id
        self.grade = grade
        self.address = address
    def __repr__(self):
        return f"{self.name}, ID:{self.student_id}, Grade:{self.grade}"

class FileAllocationTable:
    def __init__(self):
        self.Table = { }
    def add_record(self, student_id, block_index, length):
        self.table[student_id] = (block_index, length)
    def remove_record(self, student_id):
        if student_id in self.table:
            del self.table[student_id]
    def get_record_location(self, student_id):
        return self.table.get(student_id, None)

class StudentDatabase:
    def __init__(self, block_size=1):
        self.disk = [ ]
        self.fat = FileAllocationTable( )
        self.block_size = block_size
```

```
def add_student(self, student):
    block_index = len(self.disk) // self.block_size
    self.disk.append(student)
    self.fat.add_record(student.student_id, block_index, 1)
    print(f"Added: {student}")
def delete_student(self, student_id):
    record_location = self.fat.get_record_location(student_id)
    if record_location:
        block_index, _ = record_location
        self.disk[block_index] = None
        self.fat.remove_record(student_id)
        print(f" Deleted Student with ID: {student_id}")
    else:
        print(f" Student with ID: {student_id} not found.")
def update_student(self, student_id, new_student):
    record_location = self.fat.get_record_location(student_id)
    if record_location:
        block_index, _ = record_location
        self.disk[block_index] = new_student
        self.fat.add_record(new_student.student_id, block_index, 1)
        print(f" Updated Student with ID: {student_id} to {new_student
    else:
        print(f"Student with ID: {student_id} not found.")
def display_students(self):
    print(" Current Students Records:")
```

```
    for record in self.disk:
        if record:
            print(record)

def calculate_disk_space(self):
    total_space = len(self.disk) * self.block_size
    used_space = sum(1 for record in self.disk) * self.block_size
    print(f"Total disk space : {total_space} units")
    print(f"used disk space: {total_space} units")
    print(f"Free disk space : {total_space - used_space} units")

database = StudentDatabase()
Student_a = StudentRecord("John Doe",101,10,"123 main street")
Student_b = StudentRecord("Jane smith",102,11,"456 Elm Street")
Student_c = StudentRecord("Michael Brown" 103,9,"789 Oak Avenue")
database.add_student(Student_a)
database.add_student(Student_b)
database.add_student(Student_c)
database.display_students()
Student_updated = StudentRecord("John Doe", 101,10,"321 Main Street")
database.update_student(101, Student_a_updated)
database.display_students()
database.delete_student(102)
database.display_students()
database.calculate_disk_space()
```

Output: Executed.

Aim: Managing medical records in a hospital information system.

Code:

```
class patientRecord:
    def __init__(self, name, age, medical_id, address):
        self.name = name
        self.age = age
        self.medical_id = medical_id
        self.address = address
    def __repr__(self):
        return f"{self.name}, Age: {self.age}, Medical ID: {self.medical_id}"
class IndexBlock:
    def __init__(self):
        self.pointers = [ ]
    def add_pointer(self, block_index):
        self.pointers.append(block_index)
class IndexedFileAllocation:
    def __init__(self, block_size=1):
        self.disk = [ ]
        self.index_table = { }
        self.block_size = block_size
    def add_patient(self, patient):
        if patient.medical_id in self.index_table:
            print(f"Patient with Medical ID {patient.medical_id}.")
            return
        index_block = IndexBlock()
        self.disk.append(patient)
        index_block.add_pointer(len(self.disk) - 1)
```

ROLL NUMBER :

```
        self.index_table[patient.medical_id] = index_block
        print(f"Added: {patient}")
    def delete_patient(self, medical_id):
        if medical_id not in self.index_table:
            print(f"Patient with medical ID {medical_id} not found.")
            return
        index_block = self.index_table
        for block_index in index_block.pointers:
            self.disk[block_index] = None
        del self.index_table[medical_id]
        print(f"Deleted patient with medical ID: {medical_id}")
    def update_patient(self, medical_id, updated_patient):
        if medical_id not in self.index_table:
            print(f"Patient with medical ID: {medical_id} not found.")
            return
        index_block = self.index_table[medical_id]
        block_index = index_block.pointers[0]
        self.disk[block_index] = updated_patient
        print(f"updated patient with medical ID: {updatedpatient}")
    def retrive_patient(self, medical_id):
        if medical_id not in self.index_table:
            print(f"patient with medical ID {medical_id} not found.")
            return None
        index_block = self.index_table[medical_id]
        block_index = index_block.pointers[0]
        patient_record = self.disk[block_index]
```

```
        print(f"Retrieved: {patient_record}")
        return patient_record
    def display_patients(self):
        print("Current patient records:")
        for record in self.disk:
            if record:
                print(record)
    def calculate_disk_space(self):
        total_space = len(self.disk) * self.block_size
        used_space = sum(1 for record in self.disk) * self.block_size

emr_system = IndexedFileAllocation()
patient_a = PatientRecord("John Smith", 45, 1001, "123 Hospital Road")
patient_b = PatientRecord("Joane Doe", 32, 1002, "98 Clinic Avenue")
patient_c = PatientRecord("Michael", 58, 1003, "789 Medical Plaza")
emr_system.add_patient(patient_a)
emr_system.add_patient(patient_b)
emr_system.add_patient(patient_c)
emr_system.retrieve_patient(1001)
patient_a_updated = PatientRecord("John Smith", 46, 1001, "321 Hospital")
emr_system.update_patient(1001, patient_a_updated)
emr_system.display_patients()
emr_system.delete_patient(1002)
emr_system.display_patients()
emr_system.calculate_disk_space()
```

Output: Executed.

ROLL NUMBER :

3.3 Aim: Managing digital media files in a multimedia Application.

Code:

```
class MediaFile:
    def __init__(self, name, file_type, size):
        self.name = name
        self.file_type = file_type
        self.size = size
        self.blocks = [ ]
    def __repr__(self):
        return f"{self.name}({self.file_type}), size : {self.size}MB"
class DiskBlock :
    def __init__(self, index):
        self.index = index
        self.data = None
        self.next_block = None
class LinkedFileAllocation:
    def __init__(self, block_size=10):
        self.disk = [ ]
        self.fat = { }
        self.block_size = block_size
    def add_file(self, media_file):
        required_blocks = (media_file.size + self.block_size - 1)
        block_indices = [ ]
        for i in range(len(self.disk), len(self.disk) + req_blocks):
            self.disk.append(DiskBlock(i))
            block_indices.append(i)
```

ROLL NUMBER :

```
    for i in range(required_blocks):
        if i < required_blocks - 1:
            self.disk[block_indices[i]].next_block = block_indices[i+1]
        else:
            self.disk[block_indices[i]].next_block = None
    media_file.blocks = block_indices
    self.fat[media_file.name] = block_indices
    print(f"Added: {media_file}")
def delete_file(self, file_name):
    if file_name not in self.fat:
        print(f"File '{file_name}' not found.")
        return
    block_indices = self.fat[file_name]
    for block_index in block_indices:
        self.disk[block_index] = None
    del self.fat[file_name]
    print(f"Deleted file: {file_Name}")
def retrieve_file(self, file_name):
    if file_name not in self.fat:
        print(f"File '{file_name}' not found.")
        return None
    block_indices = self.fat[fat_file_name]
    file_blocks = [ ]
    for block_index in block_indices:
        if self.disk[block_index]:
```

```
        file_blocks.append(self.disk[block_index])
    print(f"Retrieved file '{file_name}' with blocks: {file_blocks}")
    return file_blocks

def display_files(self):
    print("Current Media files:")
    for file_name, block_indices in self.fat.items():
        print(f"{file_name}: Blocks {block_indices}")

def calculate_disk_space(self):
    total_space = len(self.disk) * self.block_size
    used_space = sum(self.block for block in self.disk if not None)
    print(f"total Disk Space : {total_space} MB")
    print(f"used Disk space : {used_space} MB")
    print(f"used free space : {total_space - used_space} MB")

multimedia_app = LinkedFileAllocation()
file_a = MediaFile("Landscape.jpg", "Image", 5)
file_b = MediaFile("Concert.mpu", "Video", 50)
file_c = MediaFile("Song.mp3", "Audio", 8)
multimedia_app.add_file(file_a)
multimedia_app.display_files()
multimedia_app.retrieve_file("Concert.mpu")
multimedia_app.delete_file("Song.mp3")
multimedia_app.display_files()
multimedia_app.calculate_disk_space()
```

Output: Executed