VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi, Karnataka – 590018

FILE STRUCTURES LABORATORY

(18ISL67)

SRI KRISHNA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

No.29, Hesaraghatta Main Road, Chimney hills,Chikkabanavara P.O.,

FILE STRUCTURES LABORATORY

Subject Code: 15ISL68	**I.A. Marks : 20**
Hours/Week: 1I + 02P	**Exam Hours: 03**
Total Hours: 40	**Exam Marks: 80**

PART - A

Design, Develop, and Implement the following programs

1. Write a program to read series of names, one per line, from standard input and write these names spelled in reverse order to the standard output using I/O redirection and pipes. Repeat the exercise using an input file specified by the user instead of the standard input and using an output file specified by the user instead of the standard output.

2. Write a program to read and write student objects with fixed length records and the fields delimited by "|". Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

3. Write a program to read and write student objects with Variable - Length records using any suitable record structure. Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

4. Write a program to write student objects with Variable - Length records using any suitable record structure and to read from this file a student record using RRN.

5. Write a program to implement simple index on primary key for a file of student objects. Implement add ( ), search ( ), delete ( ) using the index.

6. Write a program to implement index on secondary key, the name, for a file of student objects. Implement add ( ), search ( ), delete ( ) using the secondary index.

7. Write a program to read two lists of names and then match the names in the two lists using Cosequential Match based on a single loop. Output the names common to both the lists.

8. Write a program to read k Lists of names and merge them using k-way merge algorithm with k = 8.

****

Part B

Student should develop mini Project on the topics mentioned below or similar applications Document processing, transaction management, indexing and hashing, buffer management, configuration management. Not limited to these.

Course Outcomes

1.	Implement operations related to files
2.	Apply the concepts of file system to develop the given application.
3.	Evaluate performance of various file systems on given parameters.

Conduction of Practical Examination:

1. All laboratory experiments from part A are to be included for practical examination.
2. Mini project has to be evaluated for 30 Marks as per 6(b).
3. Report should be prepared in a standard format prescribed for project work.
4. Students are allowed to pick one experiment from the lot.
5. Strictly follow the instructions as printed on the cover page of answer script.
6. Marks distribution:
   a) Part A: Procedure + Conduction + Viva:10 + 35 +5 =50 Marks
   b) Part B: Demonstration + Report + Viva voce = 15+10+05 = 30 Marks
7. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

1. Write a program to read series of names, one per line, from standard input and write these names spelled in reverse order to the standard output using I/O redirection and pipes. Repeat the exercise using an input file specified by the user instead of the standard input and using an output file specified by the user instead of the standard output.

```
count=int(input("Enter the no. of names"))

outfile=input("Enter the output file name")

foutp=open(outfile,'w+')

for i in range(count):

    name=input("Enter the name:")

    foutp.write(name[::-1]+'\n')

    print(name[::-1])  # reverses string. ::-1 means it goes from beginning to end with step of -1,
so backwards

foutp.close()
```

2. Write a program to read and write student objects with fixed-length records and the fields delimited by "|". Implement pack ( ), unpack ( ),modify ( ) and search ( ) methods.

```
s=[]

class student:

    def __init__(self,usn,name,sem):

        self.usn=usn

        self.name=name

        self.sem=sem

    def display_data(self):

        print("\nUSN:"+self.usn+"\nNAME:"+self.name+"\nSEM:"+self.sem)

    def pack(self,file):

        buf=self.usn+"|"+self.name+"|"+self.sem+"|"

        if len(buf)>45:
            print("Length Exceeded")
        while len(buf)<46:
            buf+='_'
        buf+="\n"
        file.write(buf)
def unpack():
    with open("Out2.txt","r") as fp:
        for line in fp:
            fields=line.strip('_\n').split("|")[:-1]
            s.append(student(fields[0],fields[1],fields[2]))
while True:
    choice=input("1.Insert a record\n2.Search and Modify a record\n3.Exit\nEnter your choice")
    if choice=='1':
        usn=input("Enter USN")
        name=input("Enter name")
        sem=input("Enter sem")
        temp=student(usn,name,sem)
        with open("Out2.txt","a+") as fp:
            temp.pack(fp)
    elif choice=='3':
        break
    elif choice=='2':
        s=[]
        unpack()
        usn_srch=input("Enter the usn to search and modify")
        for x in s:
            if usn_srch==x.usn:
                print("Record found")
                ch=input("Select the field to modify\n1.Name\n2.Sem")
                if ch=='1':
                    newname=input("Enter the new name")
                    x.name=newname
                elif ch=='2':
                    newsem=input("Enter the new sem")
                    x.sem=newsem
        with open("Out2.txt","w+") as fp:
            for x in s:
                x.pack(fp)
    else:
        print("Invalid Input")
```

3. Write a program to read and write student objects with Variable -Length records using any suitable record structure. Implement pack ( ),unpack ( ), modify ( ) and search() methods.

```
s=[]
class student:
    def __init__(self,usn,name,sem):
        self.usn=usn
        self.name=name
        self.sem=sem
    def display_data(self):
        print("\nUSN:"+self.usn+"\nNAME:"+self.name+"\nSEM:"+self.sem)
    def pack(self,file):
        buf=self.usn+"|"+self.name+"|"+self.sem+"|"
        buf+="\n"
        file.write(buf)
def unpack():
    with open("Out3.txt","r") as fp:
        for line in fp:
            fields=line.strip('\n').split("|")[:-1]#[:-1] is needed to not include last | char
            s.append(student(fields[0],fields[1],fields[2]))
while True:
    choice=input("1.Insert a record\n2.Search and Modify a record\n3.Exit\nEnter your choice")
    if choice=='1':
        usn=input("Enter USN")
        name=input("Enter name")
        sem=input("Enter sem")
        temp=student(usn,name,sem)
        with open("Out3.txt","a+") as fp:
            temp.pack(fp)
    elif choice=='3':
        break
    elif choice=='2':
        s=[]
        unpack()
        usn_srch=input("Enter the usn to search and modify")
        for x in s:
            if usn_srch==x.usn:
```

```
            print("Record found")
            ch=input("Select the field to modify\n1.Name\n2.Sem")
            if ch=='1':
               newname=input("Enter the new name")
               x.name=newname
            elif ch=='2':
               newsem=input("Enter the new sem")
               x.sem=newsem
      with open("Out3.txt","w+") as fp:
#          fp.seek(0)
#          fp.truncate()
         for x in s:
            x.pack(fp)
   else:
      print("Invalid Input")
```

4. Write a program to write student objects with Variable – Length records using any suitable record structure and to read from this file a student record using RRN.

Program-4:

Write a program to write student objects with Variable - Length records using any suitable record structure and to read from this file a student record using RRN.

```
rrn=[-1]
cnt=0
class student:
   def __init__(self,usn,name,sem):
      self.usn=usn
      self.name=name
```

```
        self.sem=sem
    def display_data(self):
        print("\nUSN:"+self.usn+"\nNAME:"+self.name+"\nSEM:"+self.sem)
    def pack(self,file):
        global cnt
        pos=file.tell()
        buf=self.usn+"|"+self.name+"|"+self.sem+"|"
        buf+="\n"
        file.write(buf)
        cnt+=1
        rrn.append(pos)
def unpack(pos):
    with open("record.txt","r") as fp:
        fp.seek(pos)
        line=fp.readline()
        fields=line.strip('\n').split("|")[:-1]
        s1=student(fields[0],fields[1],fields[2])
        s1.display_data()
def find_rrn():
    global cnt,pos,rrn
    pos=0
    try:#try is needed since otherwise, if record is empty, it will give an error
        with open("record.txt","r+") as fp:
            line = fp.readline()
            while line:
                rrn.append(pos)
                pos=fp.tell() #returns the location of the next line
                cnt+=1
            line = fp.readline()
    except:
        pass
'''def find_rrni():
    global cnt,pos,rrn
    pos=0
    i=1
    try:
        with open("record.txt","r+") as fp:
            for line in fp:
                rrn.append(pos)
                pos+=len(line)+i
                cnt+=1
                i+=1
                #print(line)
    except:
        pass'''
find_rrn()
while True:
    choice=input("1.Insert a record\n2.Search for a record using RRN\n3.Exit\nEnter your choice")
    if choice=='1':
        usn=input("Enter USN")
        name=input("Enter name")
        sem=input("Enter sem")
        temp=student(usn,name,sem)
        with open("record.txt","a+") as fp:
            temp.pack(fp)
    elif choice=='3':
        break
    elif choice=='2':
        print(cnt)
        rrn_srch=int(input("Enter the RRN to be found"))
        if rrn_srch>cnt or rrn_srch<0:
            print("Invalid RRN")
            continue
        print("Record found")
        pos=rrn[rrn_srch]
        with open("record.txt","r") as fp:
            unpack(pos)
    else:
        print("Invalid Input")
```

5. Write a program to implement simple index on primary key for a file of student objects. Implement add ( ), search ( ), delete ( ) using the index.

```
i1 = []
cnt = -1
class student:
    def __init__(self,usn,name,sem):
        self.usn=usn
        self.name=name
        self.sem=sem
    def pack(self,file):
        global cnt,i1
        pos=file.tell()
        buf=self.usn+"|"+self.name+"|"+self.sem+"|"
        buf+="\n"
        file.write(buf)
        cnt+=1
        i1.append(index(self.usn,pos))
        i1.sort(key = lambda x:x.usn)


class index:
    def __init__(self,usn,addr):
        self.usn=usn
        self.addr=addr


def create_index():
    global cnt, pos, i1
    pos = 0
    try:
        with open("record5.txt","r") as fp:
            line = fp.readline()
            while line:
                if line.startswith('*') or len(line) == 0:#if the record is deleted, dont add to index and read the
next record
                    line = fp.readline()
                    pos = fp.tell()
                else:
                    fields=line.strip('\n').split("|")[:-1]
                    i1.append(index(fields[0], pos))
                    pos = fp.tell()
                    cnt += 1
                line = fp.readline() #needed since when we delete, extra blank line is present at the end of the
file
        i1.sort(key = lambda x:x.usn)#sort index list based on usn
```

```
#        for y in i1:#to check if i1 is correct when prog. closedand opened and if it prints in sorted order
#            print(y.usn,y.addr)
    except:
        pass

def find_index(usn_srch):
    ind = -1
    for i in range(cnt+1):
        if i1[i].usn == usn_srch:
            ind = i
    return ind

def search():
    global i1
    usn_srch = input("Enter the USN of the student to be found")
    ind = find_index(usn_srch)
    if ind == -1:
        print('Record not found')
    else:
        print('Record found\nUSN|Name|Sem')
        with open("record5.txt","r") as fp:
            fp.seek(i1[ind].addr)
            line = fp.readline()
            print(line.strip('\n'))

def delete():
    global i1,cnt
    usn_srch = input("Enter the USN of the student to be deleted")
    ind = find_index(usn_srch)
    if ind == -1:
        print('Record not found')
    else:
        print('Record deleted')
        print(i1[ind].addr)
        with open("record5.txt","r+") as fp:
            fp.seek(i1[ind].addr)
            fp.write('*')#if a record has *=>means it is deleted
        i1.pop(ind)#remove that element from the index array
        cnt -= 1#reduce the count of the no. of elements
create_index()#has to be called when prog. starts
while True:
    choice = int(input('1.Add a record\n2.Search for a record\n3.Delete a record\n4.Exit\nEnter choice'))
    if choice == 1:
        usn=input("Enter USN")
        name=input("Enter name")
        sem=input("Enter sem")
        s1=student(usn,name,sem)
        with open("record5.txt","a+") as fp:
            s1.pack(fp)
    elif choice == 2:
        search()
    elif choice == 3:
        delete()
    else:
        break
```

6. Write a program to implement index on secondary key, the name, for a file of student objects. Implement add ( ), search ( ), delete ( ) using the secondary index.

```
i2 = []

cnt = -1

class student:

    def __init__(self,usn,name,sem):

        self.usn=usn

        self.name=name

        self.sem=sem

    def pack(self,file):

        global cnt,i2

        pos=file.tell()

        buf=self.usn+"|"+self.name+"|"+self.sem+"|"

        buf+="\n"

        file.write(buf)

        cnt+=1

        i2.append(sec_index(self.usn,self.name,pos))

        i2.sort(key = lambda x:x.name)

class sec_index:

    def __init__(self,usn,name,addr):

        self.usn=usn

        self.name=name

        self.addr=addr

def create_index():

    global cnt, pos, i2

    pos = 0

    try:

        with open("record6.txt","r") as fp:

            line = fp.readline()
        while line:
            if line.startswith('*') or len(line) == 0:#if the record is deleted, dont add to index and read the
next record
                line = fp.readline()
                pos = fp.tell()
            else:
                fields=line.strip('\n').split("|")[:-1]
                i2.append(sec_index(fields[0], fields[1], pos))
                pos = fp.tell()
                cnt += 1
            line = fp.readline() #needed since when we delete, extra blank line is present at the end of the
file
        i2.sort(key = lambda x:x.name)#sort index list based on usn
#       for y in i2:#to check if i1 is correct when prog. closedand opened and if it prints in sorted order
#           print(y.usn,y.addr)
    except:
        pass
def find_sec(name_srch):
    global find_cnt,found,indexnums
    find_cnt = 0
    indexnums = []
    found = []
    for ind, i in enumerate(i2):#enumerate=>to obtain index and val in same loop
        if i.name == name_srch:
            found.append(i)
            indexnums.append(ind)#array of the indices=> will be needed in delete
            find_cnt += 1


def search():
    global i2, found, find_cnt
    name_srch = input("Enter the name of the student to be searched")
    find_sec(name_srch)
    if find_cnt == 0:
        print('Record not found')
    elif find_cnt == 1:
        print('One record found')
        ch = 0
    else:
        print('Multiple records found')
        for i in range(find_cnt):
            print(i, "USN="+found[i].usn)
        ch = int(input('Enter choice:'))
        if ch > find_cnt:
            print('Invalid range')
            return
    print('USN|Name|Sem')
    with open("record6.txt","r") as fp:
        fp.seek(found[ch].addr)
        line = fp.readline()
        print(line.strip('\n'))
def delete():
    global i2, find_cnt, found, indexnums, cnt
    name_srch = input("Enter the name of the student to be deleted")
    find_sec(name_srch)
    if find_cnt == 0:
        print('Record not found')
    elif find_cnt == 1:
        print('One record found')
        ch = 0
    else:
        print('Multiple records found')
        for i in range(find_cnt):
            print(i, "USN="+found[i].usn)
        ch = int(input('Enter choice:'))
        if ch > find_cnt:
            print('Invalid range')
            return
    print('Record deleted')
    with open("record6.txt","r+") as fp:#r+ means read and write. stream positioned at beginning of
file=>so we can use seek unlike a+
        fp.seek(found[ch].addr)
        fp.write('*')#if a record has *=>means it is deleted
    i2.pop(indexnums[ch])#remove that element from the index array
    cnt -= 1#reduce the count of the no. of elements
create_index()#has to be called when prog. starts
while True:
    choice = int(input('1.Add a record\n2.Search for a record\n3.Delete a record\n4.Exit\nEnter choice'))
    if choice == 1:
        usn=input("Enter USN")
        name=input("Enter name")
        sem=input("Enter sem")
        s1=student(usn,name,sem)
        with open("record6.txt","a+") as fp:
            s1.pack(fp)
    elif choice == 2:
        search()
    elif choice == 3:
        delete()
    else:
        break
```

7. Write a program to read two lists of names and then match the names in the two lists using Consequential Match based on a single loop. Output the names common to both the lists.

```
with open("list1.txt") as fp1:
    list1 = fp1.read().split("\n")
with open("list2.txt") as fp2:
    list2 = fp2.read().split("\n")
count1 = len(list1)
count2 = len(list2)
i = 0
j = 0
list3=[]
list1.sort()
list2.sort()
while i <count1 and j <count2:
    if list1[i] < list2[j]:
        i += 1
    elif list1[i] > list2[j]:
        j += 1
    else:
        list3.append(list1[i])
        i += 1
        j += 1
with open("output.txt",'w+') as fp3:
    for name in list3:
        fp3.write(name+'\n')
```

Program 8: **Write a program to read k Lists of names and merge them using k way merge algorithm with k = 8.**

```
lst = []


#sort function
for i in range(4):
    with open('n'+str(i+1)+'.txt') as fp:
        list1 = fp.read().split('\n')
    list1.sort()
    with open('ns'+str(i+1)+'.txt','w') as fp:#to sort and store the files
        for x in list1:
            fp.write(x+'\n')
files = ["ns1.txt", "ns2.txt","ns3.txt","ns4.txt"]#list of all the sorted files
fhandler = []#list of the active filehandlers
for f in files:
    fhandler.append(open(f,'r'))
lines = []#read the first name from each file and store in list
for fh in fhandler:
    lines.append(fh.readline())
while len(fhandler) > 0:#while there are still files to be read
    smallest = min(lines)#find the smallest name
    smallestposition = lines.index(smallest)#find the file which has the smallest name
    lst.append(smallest)#append the smallest name in o/p list
    lines[smallestposition] = fhandler[smallestposition].readline()#and keep reading from that file
    if lines[smallestposition] == "":#if the file has been read completely
```

```
        fhandler.pop(smallestposition)#pop that file handler as it isnt needed
        lines.pop(smallestposition)#pop it from the lines file
print('Merged List:')
for names in sorted(set(lst)):#since set is unordered
    print(names.strip('\n'))
```

SAMPLE VIVA QUESTIONS

1. Define file structure
2. What is seeking? How is it supported in C streams and C++ streams?
3. List fundamental file processing operations
4. What is physical and logical device
5. What is buffer management?
6. Define : rotational delay, seek time, transfer time.
7. Define record access and record structures
8. Define field structure
9. What are the common methods of adding structures to files to maintain identity of fields
10. What is data compression?
11. Mention the types of placement strategies
12. Explain key sort algorithm
13. What are the basic operations on entry sequenced files?
14. What is inverted list? What are its advantages?
15. Explain k-way merge concept
16. Explain the different types of co-sequential processing in UNIX
17. What is multilevel indexing?
18. Define B-tree
19. List the different operations required to maintain an indexed file
20. Explain the different methods of secondary index structures
21. What is AVL tree and paged binary tree?
22. Wrt B-tree what is worst case search depth and deletion, merging?
23. Explain the structure of indexed sequential file
24. What are the similarities between B-tree and B+ trees
25. What are the differences between B-tree and B+ trees?
26. What is matching and merging?
27. What is primary index, secondary index and selective index?
28. What are the difficulties associated with secondary index structures?
29. How does extensible hashing works?
30. Define hashing.

31. What are the different ways by which portability can be achieved in files?
32. Differentiate between internal and external fragmentation
33. What are the different ways by which fragmentation can be minimized?
34. What is fragmentation?
35. What are the problems associated with binary search on secondary storage?
36. In what way is an AVL tree better than a simple binary search tree?
37. List out considerations for block size
38. What are the different costs for disk access?
39. Define index
40. What are the properties of B-tree?
41. What are the different techniques available for data compression?
42. Describe the parameters for the functions OPEN,READ and WRITE.
43. What is the difference between dynamic hashing and linear hashing?