LABORATORY MANUAL

17ISL68 - File Structures Laboratory

2019-20

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING
ATRIA INSTITUTE OF TECHNOLOGY
Adjacent to Bangalore Baptist Hospital
Hebbal, Bengaluru-560024

SYLLABUS

SEMESTER – VI

Subject Code 17ISL68 **IA Marks 40**
Exam Marks 60 **Exam Hours 03**

PART A

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 Consequential 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

Mini Project -- 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.

CONTENTS

1) Write a program to read a 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.

I/O redirection

Operating systems provide shortcuts for switching between standard I/O (stdin and stdout) and regular file I/O. I/O redirection is used to change a program so it writes its output to a regular file rather than to stdout.

• In both DOS and UNIX, the standard output of a program can be redirected to a file with the > symbol.

• In both DOS and UNIX, the standard input of a program can be redirected to a file with the < symbol.

• The notations for input and output redirection on the command line in Unix are

```
<file   redirect stdin to "file"
>file   redirect stdout to "file"
```

• **Example:** List.exe > myfile

The output of the executable file is redirected to a file called "myfile"

Pipe

• Piping: using the output of one program as input to another program. A connection between standard output of one process and standard input of a second process.

• In both DOS and UNIX, the standard output of one program can be piped (connected) to the standard input of another program with the | symbol.

• **Example:** program1 | program2

• Output of program1 is used as input for program2

File I/O: perform output and input of characters to or from files

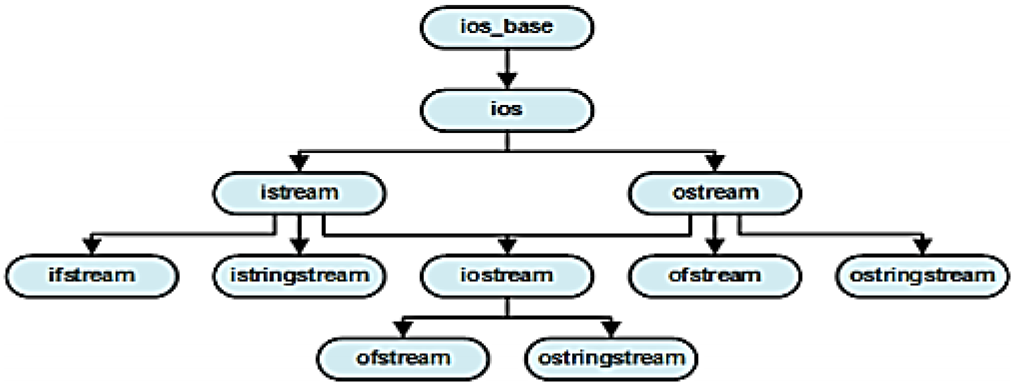
Standerd I/O:

• standard streams are preconnected input and output channels between a computer program and its environment (typically a text terminal) when it begins execution. The three I/O connections are called standard input (stdin), standard output (stdout) and standard error (stderr).

Fstream

• fstream provides an interface to read and write data from files as input/output streams. The file to be associated with the stream can be specified either as a parameter in the constructor or by calling member open.

• After all necessary operations on a file have been performed, it can be closed (or disassociated) by calling member close. Once closed, the same file stream object may be used to open another file.



Function to open a file:

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to open a file. An open file is represented within a program by a stream object (an instantiation of one of these classes, in the previous example this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function open():

open (filename, mode);

Where filename is a null-terminated character sequence of type const char * (the same type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.
ios::trunc	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

Function to Closing the Files

To disassociate a logical program file from a physical system file.

• **Prototypes:**

int close (int Handle);

• **Example:**

close (Input);

Getline Function: Extracts characters from specified location and stores them into str until the delimitation character delim is found or length equal to size.

• **prototype**

fstream str;

Str.getline (istream& is, int size, char delim);

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.

Fixed length record

A record which is predetermined to be the same length as the other records in the file.

Record 1	Record 2	Record 3	Record 4	Record 5

• The file is divided into records of equal size.
• All records within a file have the same size.
• Different files can have different length records.
• Programs which access the file must know the record length.
• Offset, or position, of the nth record of a file can be calculated.
• There is no external overhead for record separation.
• There may be internal fragmentation (unused space within records.)
• There will be no external fragmentation (unused space outside of records) except for deleted records.
• Individual records can always be updated in place

Delimited Variable Length Fields

Record 1	|	Record 2	|	Record 3	|	Record 4	|	Record 5

• The fields within a record are followed by a delimiting byte or series of bytes.
• Fields within a record can have different sizes.
• Different records can have different length fields.
• Programs which access the record must know the delimiter.
• The delimiter cannot occur within the data.
• If used with delimited records, the field delimiter must be different from the record delimiter.
• There is external overhead for field separation equal to the size of the delimiter per field.
• There should be no internal fragmentation (unused space within fields.)

Pack():

This method is used to group all the related field values of particular record taken by the application in buffer.

Unpack():

This method is used to ungroup all the related field values of particular record taken from the file in buffer.

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.

Variable length record
A record which can differ in length from the other records of the file.

• delimited record
A variable length record which is terminated by a special character or sequence of characters.

• delimiter
A special character or group of characters stored after a field or record, which indicates the end of the preceding unit.

• The records within a file are followed by a delimiting byte or series of bytes.
• The delimiter cannot occur within the records.
• Records within a file can have different sizes.
• Different files can have different length records.
• Programs which access the file must know the delimiter.
• Offset, or position, of the nth record of a file cannot be calculated.
• There is external overhead for record separation equal to the size of the delimiter per record.
• There should be no internal fragmentation (unused space within records.)
• There may be no external fragmentation (unused space outside of records) after file updating.
• Individual records cannot always be updated in place.

Program:

```
#include<iostream>
#include<fstream>
#include<string.h>
#define SIZE 55
using namespace std;
char buffer[SIZE + 1];


class Student {
        char usn[15];
        char name[20];
        char sem[5];
        char marks[10];
        public:
        void getData();
```

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(relative record number)

• RRN is an ordinary number that gives the distance of current record from first record. Using RRN, Direct access allows individual records to be read from different locations in the file without reading intervening records.

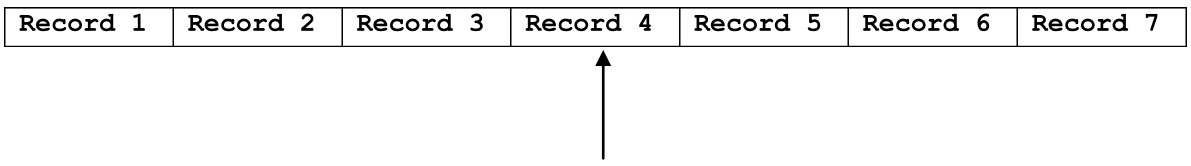
• When we are using fixed length record, we can calculate the byte offset of each record using the fallowing formula

• ByteOffset = (RRN - 1) × RecLen

o RRN: relative record number(starts fron 0)

o RecLen: size of fixed length record

Direct Access

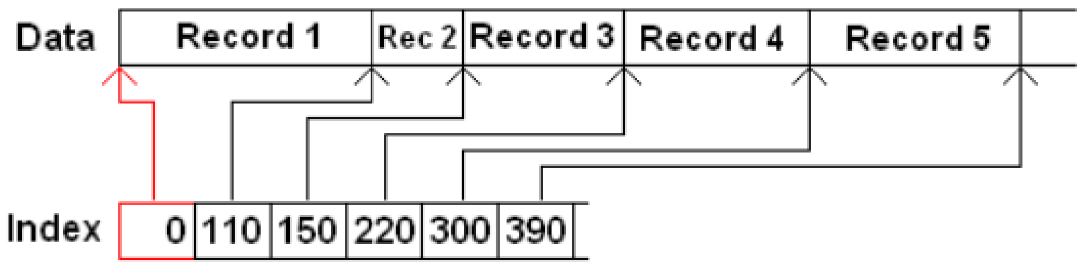


Program:

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;
class student
{
  public:
        string USN;
        string Name;
        string Branch;
        int Semester;
        string buffer;
        int count;

        int rrn_list[100];

        void read_data();
        void pack();
```

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

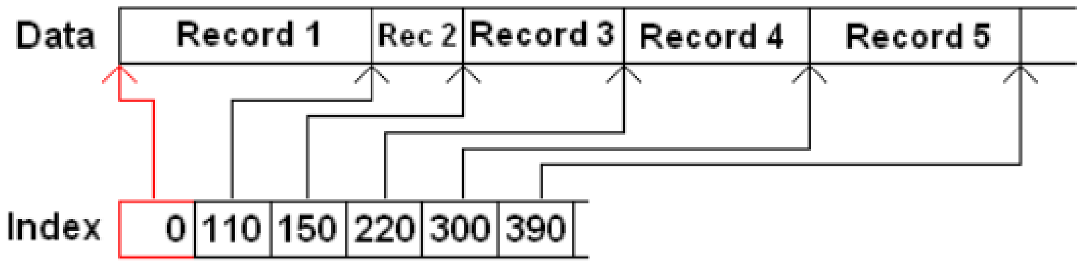
Index
A structure containing a set of entries, each consisting of a key field and a reference field, Which is used to locate records in a data file.
Key field
The part of an index which contains keys.
Reference field
The part of an index which contains information to locate records.
• An index imposes order on a file without rearranging the file.
• Indexing works by indirection.
Simple Index for Entry-Sequenced Files
Simple index
• An index in which the entries are a key ordered linear list. Simple indexing can be useful when the entire index can be held in memory. Changes (additions and deletions) require both the index and the data file to be changed.
• Updates affect the index if the key field is changed, or if the record is moved. An update which moves a record can be handled as a deletion followed by an addition.



Primary key
A primary key is a special relational database table column (or combination of columns) designated to uniquely identify all table records. A primary key's main features are: It must contain a unique value for each row of data. It cannot contain null values.
Program:
```
#include<iostream>

#include<fstream>

#include<string>

#include<sstream>

using namespace std;

class primary_index

{

        public:
```

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.

Index
A structure containing a set of entries, each consisting of a key field and a reference field, Which is used to locate records in a data file.
Key field
The part of an index which contains keys.
Reference field
The part of an index which contains information to locate records.
• An index imposes order on a file without rearranging the file.
• Indexing works by indirection.
Simple Index for Entry-Sequenced Files
Simple index
• An index in which the entries are a key ordered linear list. Simple indexing can be useful when the entire index can be held in memory. Changes (additions and deletions) require both the index and the data file to be changed.
• Updates affect the index if the key field is changed, or if the record is moved. An update which moves a record can be handled as a deletion followed by an addition.



Secondary key
A secondary key is made on a field that you would like to be indexed for faster searches. A table can have more than one secondary key
Program:

```
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
using namespace std;
class secondary_index{

        public:
                string Name_list[100];
                int Address_list[100];
                int count;
```

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.

Consequential operations

Operations which involve accessing two or more input files sequentially and in parallel, resulting in one or more output files produced by the combination of the input data.

Considerations for Consequential Algorithms

•Initialization - What has to be set up for the main loop to work correctly?
•Getting the next item on each list - This should be simple and easy, from the main algorithm.
•Synchronization - Progress of access in the lists should be coordinated.
•Handling End-Of-File conditions - For a match, processing can stop when the end of any list is reached.
•Recognizing Errors - Items out of sequence can "break" the synchronization.

Matching Names in Two Lists Match

The process of forming a list containing all items common to two or more lists.

Consequential Match Algorithm

•Initialize (open the input and output files.)
•Get the first item from each list.
•While there is more to do:
Compare the current items from each list.
If the items are equal,
Process the item.
Get the next item from each list.
Set more to true iff none of these lists is at end of file.
If the item from list A is less than the item from list B,
Get the next item from list A.
Set more to true iff list A is not at end-of-file.
If the item from list A is more than the item from list B,
Get the next item from list B.
Set more to true iff list B is not at end-of-file.
• Finalize (close the files.)

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

Merge
The process of forming a list containing all items in any of two or more lists.
K-way merge
A merge of order k
Order of a merge
The number of input lists being merged.

- If the distribution phase creates k runs, a single k-way merge can be used to produce the final sorted file.
- A significant amount of seeking is used by a k-way merge, assuming the input runs are on the same disk.

Program:
```
#include<iostream>
#include<fstream>
#include<string.h>
using namespace std;
// Record specification
class record
{
        public: char name[20];
        char usn[20];
}rec[20];
int no;
fstream file[8];
//The first 8 files
char fname[8][8] = {"l.txt","2.txt","3.txt","4.txt","5.txt","6.txt","7.txt","8.txt"};
void merge_file(char* file1, char* file2, char* filename)
{
        record recrd[20];
        int k; k=0;
        fstream f1,f2;
        f1.open(file1,ios::in);   //open the first file
        f2.open(file2,ios::in);   //open the second file
```

VIVA QUESTIONS

1. What is FILE STRUCTURE?
2. What are the main goals of FILE STRUCTURE?
3. What is a physical file?
4. What is a logical file?
5. Difference between physical and logical file?
6. What is the syntax of open function?
7. Define Flags.
8. List different types of flag and their operations.
9. What do you mean by pmode? Explain.
10. Define read function with syntax.
11. Define Write function with syntax.
12. What do you mean by source file?
13. What do you mean by destination file?
14. What are source and destination addresses?
15. Define File descriptor.
16. Explain seeking with its syntax.
17. Difference between seekg and seekp.
18. Define byte offset.
19. What is origin in seek function?
20. Explain the following commands.

a)cat	d)mv	g)ls
b)tail	e)rm	h)mkdir
c)cp	f)chmod	i)rmdir

21. Explain different field structures.
22. Explain different record structures.
23. What is inheritance?
24. Explain primary key with an Example.
25. Explain Secondary key with an Example.
26. What do you mean by canonical form?
27. What is Sequential search?
28. How can we improve the sequential search performance?
29. What is the order of sequential search?
30. When will you say that the sequential search is good?
31. Give some unix tools for sequential processing.

32. What is Direct Access?
33. Explain the order of Direct Access.
34. RRN stands for?
35. Define RRN.
36. How can we Calculate byte offset using RRN?
37. What do you mean by Standard I/O?
38. What are header records?
39. What is data compression?
40. Advantages of data compression.
41. Different techniques of data compression.
42. What is run-length encoding?
43. What do you mean by reclaimation of unused space?
44. What is linked-list?
45. What is need of using linked list?
46. What is stack?
47. Explain the need of using stack?
48. What do you mean by available list?
49. What “-1” indicates in avail list?
50. How you can reclaim the unused space?
51. What is storage fragmentation?
52. What do you mean by internal and external Fragmentation?
53. What are the different placement strategies? Explain.
54. Limitations of Binary search.
55. What is keysorting?
56. What is tagsort?
57. Define pinned record.
58. Define Redundancy Reduction.
59. What do you mean by indexing?
60. What are the Operations required to maintain the indexed file?
61. What are inverted lists?
62. Define selective indexes.
63. Define Binding.
64. Define Co-sequential search.
65. Explain the logic K-Way merge.
66. AVL stands for?
67. What do you mean by paged Binary trees?
68. Define multilevel indexing.

69. What are B-trees?
70. Explain Splitting.
71. Explain the logic of redistribution.
72. Explain how you are deleting the nodes in B trees.
73. Explain the logic for merging the nodes in B trees.
74. Define B* trees.
75. Explain Virtual B-trees.
76. What is LRU replacement?
77. What do you mean by replacement based on page height? Explain.
78. What is order of a B-tree?
79. Define page index.
80. What are blocks?
81. What are separators?
82. Define prefix.
83. Define B+ tree.
84. Define hashing.
85. Define hashing function.
86. Explain the advantages and disadvantages of hashing.
87. Explain any simple hashing algorithm.
88. Define Packing Density.
89. What is Progressive overflow?
90. Define search length.
91. What are buckets?
92. What are tombstones?
93. Define double hashing.
94. Define chained progressive overflow.
95. What do you mean by chaining with a separate overflow Area?
96. Define scatter tables.
97. What is a pattern of record access?
98. What do you mean by home address?
99. What is Mid-square method?
100. Define Extendible hashing?
101. Define buddy buckets?
102. Define linear hashing.
103. Explain dynamic hashing.

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Vision:

To be centre of excellence in Information Science and Engineering equipping students with top notch competencies in the domain of information technology.

Mission:

- Promote best teaching – learning , research, innovation and also instill professional ethics, cultural values and environmental awareness among the students
- Establishing learning ambience with best infrastructure facilities