



||Jai Sri Gurudev||
Sri AdichunchanagiriShikshana Trust(R)



SJB Institute of Technology

No. 67, BGS Health & Education City, Dr. Vishnuvardhan Road
Kengeri, Bangalore – 560 060



Department of Information Science and Engineering

File Structures Laboratory with mini project Manual

[18ISL67]

VI SEMESTER – B. E



Staff Name:	Sridevi G M and Vishruth B Gowda		
Section:	A & B	Batch:	A1, A2, A3, B1, B2 and B3

PREFACE

File Structures deals with organization of data on secondary storage devices. It is a combination of representations for data in files and of operations for accessing the data.

The current manual is prepared to equip students with information required to conduct experiments necessary to understand and implement basic record structuring techniques and file organization techniques like indexing and co-sequential processing. The manual gives necessary information about basic functions and classes used to perform various file operations.

This lab also comprises of implementation of file organization techniques for mini project to help students learn to handle a project, documentation for report and improves communication skills of the students.

SJB INTITUTE OF TECHNOLOGY

Institution's Vision

To become a recognized technical education centre with a global perspective

Institution's Mission

To provide learning opportunities that foster students ethical values, intelligent development in science & technology and social responsibility so that they become sensible and contributing members of the society.

Department of Information Science and Engineering

Department Vision

We envision our department as a catalyst for developing educated, engaged and employable individuals whose collective energy will be the driving force for prosperity and the quality of life in our diverse world.

Department Mission

Our mission is to provide quality technical education in the field of information technology and to strive for excellence in the education by developing and sharpening the intellectual and human potential for good industry and community.

PROGRAM EDUCATIONAL OBJECTIVES (PEO'S)

Graduates will -

- Possess expertise in problem solving, design and analysis, technical skills for a fruitful career accomplishing professional and social ethics with exposure to modern designing tools and technologies in Information Science and Engineering.
- Excel in communication, teamwork and multipledomains related to engineering issues accomplishing social responsibilities and management skills.
- Outclass in competitive environment through certification courses, gaining leadership qualities and progressive research to become successful entrepreneurs.

PROGRAM SPECIFIC OUTCOMES (PSO'S)

Graduates will be able to -

1. **PSO1:** Apply the Knowledge of Information Science to develop software solutions in current research trends and technology.
2. **PSO2:** Create Social awareness & environmental wisdom along with ethical responsibility to lead a successful career and sustain passion using optimal resources to become an Entrepreneur.

PROGRAM OUTCOMES-PO's**Engineering graduates will be able to:**

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

University Syllabus

Course Title: File Structures Laboratory with Mini Project As per Choice Based Credit System (CBCS) scheme SEMESTER:VII			
Subject code	17ISL68	IA Marks	40
Number of lecture hours/week	01I+02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03
Credits -02		Total Marks-100	
Course objectives: This course will enable students, 1. To learn basic file operations like open, read, write, close etc. 2. To learn different methods for field and record organization 3. To implement different file organization techniques like indexing. 4. To apply file operations for consequential processing.			
Sl.No.	Experiment	RBT	
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.	L1,L3	
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.	L1,L2,L3	
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.	L1,L2,L3	
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.	L1,L2,L3	
5	Write a program to implement simple index on primary key for a file of student objects. Implement add (), search (), delete () using the index.	L1,L2,L3	
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.	L1,L2,L3	

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.	L1,L2,L3
8	Write a program to read k Lists of names and merge them using k-way merge algorithm with k = 8.	L1,L2,L3
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.		
Course Outcomes: After studying this course, students will be able to: <ol style="list-style-type: none"> 1. Implement operations related to files 2. Apply the concepts of file system to produce the given application. 3. Evaluate performance of various file systems on given parameters. 		
Program Objectives: <ol style="list-style-type: none"> 1. Evaluation of the test results and assesses the impact on water and waste water treatment. 2. Train student to undertake student project work in 8th semester in the field of environmental engineering. 		
Question Paper Pattern: <ol style="list-style-type: none"> 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: 09 + 42 +09 =60 Marks b) Part B: Demonstration + Report + Viva voce = 20+14+06 = 40 Marks 7. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero. 		

COURSE OUTCOMES

On successful completion of this course students will be able to,

CO's	COURSE OUTCOMES	PO's and PSO's MAPPING
CO1	Implement various operations such insert, search, delete and modify on files.	PO2, PO3,PO5,PO9, PO10,PO11,PO12 / PSO1
CO2	Design and develop record organization techniques on files.	PO2, PO3,PO5,PO9, PO10,PO11,PO12 / PSO1
CO3	Design and develop indexing techniques on files.	PO2, PO3,PO5,PO9, PO10,PO11,PO12 / PSO1

CO4	Design and develop co-sequential processing and merging concept for files.	PO2, PO3,PO5,PO9, PO10,PO11,PO12 / PSO1
-----	--	---

General Instructions for the Laboratory

Do's

- It is mandatory for all the students to attend all practical classes & complete the experiments as per syllabus.
- Students should strictly follow the lab timings, dress code with Apron & ID cards.
- Should maintain a neat observation book.
- Study the theory and logic before executing the program.
- Submit the completed lab records of executed programs and update the index book in every lab session.
- Should prepare for viva questions regularly.
- Handle the computer systems carefully.
- Maintain discipline and silence in the lab.

Don'ts

- Should not take Bags and Mobile phones into the Laboratory.
- Do not wear footwear inside the Laboratory
- Systems & Components should be handled carefully failing to which penalty will be imposed.
- Do not switch off the system abruptly.
- Should not chew gum or eat in the lab.

LAB INDEX SHEET

Sl.No.	Experiment Description	Page No.
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.	

INTRODUCTION

All the programs are executed in Turbo C++ 3.0 version. To support this version of software, create a text file in DOS shell only to execute programs.

Creating a file in DOS shell:

In File Menu - > Go to DOS shell -> it switches to command prompt ->

Type edit filename.txt

Example : C:\tc>edit file1.txt

Include the contents of the file->save the file -> quit

Type exit command in DOS shell to switch from DOS shell to turbo C++ shell.

Basic functions and classes used in implementing the file structures programs:

C++ provides the following classes to perform output and input of characters to/from files:

ofstream: Stream class to write on files

ifstream: Stream class to read from files

fstream: Stream class to both read and write from/to files.

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 (i.e., an object 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);
```

Example:

```
fstream fp1; // creates fp1 object of type fstream class
```

```
fp1.open(fin,ios::in); // opens the file in read mode
```

Where filename is a string representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

Mode	Description
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, 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.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

Each of the open member functions of classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in ios::out

get and put stream positioning

All I/O streams objects keep internally -at least- one internal position: ifstream, like istream, keeps an internal get position with the location of the element to be read in the next input operation. ofstream, like ostream, keeps an internal put position with the location where the next element has to be written. Finally, fstream, keeps both, the get and the put position, like iostream. These internal stream positions point to the locations within the stream where the next reading or writing operation is performed. These positions can be observed and modified using the following member functions:

tellg() and tellp()

Syntax:

streampos tellg();

Get position in input sequence

Returns the position of the current character in the input stream.

Syntax:

streampos tellp();

Get position in output sequence

Returns the position of the current character in the output stream.

These two member functions with no parameters return a value of the member type streampos, which is a type representing the current get position (in the case of tellg) or the put position (in the case of tellp).

seekg() and seekp()

These functions allow to change the location of the get and put positions. Using this prototype, the stream pointer is changed to the absolute position position (counting from the beginning of the file).

seekg (offset, direction);

`seekp (offset, direction);`

Using this prototype, the get or put position is set to an offset value relative to some specific point determined by the parameter direction. offset is of type streamoff. And direction is of type seekdir, which is an enumerated type that determines the point from where offset is counted from, and that can take any of the following values:

direction	Description
<code>ios::beg</code>	offset counted from the beginning of the stream
<code>ios::cur</code>	offset counted from the current position
<code>ios::end</code>	offset counted from the end of the stream

Example:

```
fp.seekp(-27,ios::cur); // put position is set to -27 positions from current position.  
                        // - (minus) means backward direction  
fp.seekg(17.offset,ios::beg); //get position is set to 17 position from beginning of the stream
```

Checking file state flags

In addition to good, which checks whether the stream is ready for input/output operations, other member functions exist to check for specific states of a stream (all of them return a bool value):

`bad()`

Returns true if a reading or writing operation fails. For example, in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

`fail()`

Returns true if error occurs while performing read/write operation.

`eof()`

Returns true if a file open for reading has reached the end.

`good()`

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true. Note that good and bad are not exact opposites (good checks more state flags at once).

Closing a file

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function close. This member function takes flushes the associated buffers and closes the file:

```
myfile.close();
```

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes. In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function close.

Get line function

Extracts characters from the stream as unformatted input and stores them into s as a c-string, until either the extracted character is the delimiting character, or n characters have been written to s (including the terminating null character). The delimiting character is the newline character ('\n') for the first form, and delim for the second: when found in the input sequence, it is extracted from the input sequence, otherwise discarded and not written to s.

Syntax:

```
istream& getline (char* s, streamsize n );  
istream& getline (char* s, streamsize n, char delim );
```

Example:

```
char usn[30];  
fp.getline(usn,30,""); // extracts 30 characters from file and stores in usn
```

strtok function

Split string into tokens

Syntax:

```
char * strtok ( char * str, const char * delimiters );
```

Example:

```
char *t;  
t= strtok(buffer,"");  
cout<<"USN:"<<t<<endl;  
t=strtok(NULL,"");
```

A sequence of calls to this function splits str into tokens, which are sequences of contiguous characters separated by any of the characters that are part of delimiters. On a first call, the function expects a C string as argument for str, whose first character is used as the starting location to scan for tokens. In subsequent calls, the function expects a null pointer and uses the position right after the end of last token as the new starting location for scanning. To determine the beginning and the end of a token, the function first scans from the starting location for the first character not contained in delimiters (which becomes the beginning of the token). And then scans starting from this beginning of the token for the first character contained in delimiters, which becomes the end of the token. The scan also stops if the terminating null character is found. This end of the token is automatically replaced by a null-character, and the beginning of the token is returned by the function. Once the terminating null character of str is found in a call to strtok, all subsequent calls to this function (with a null pointer as the first argument) return a null pointer.

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 input file specified by the user instead of the standard input and output file specified by the user instead of the standard output.*

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<string.h>
#include<process.h>

void main()
{
    fstream fp1,fp2;
    char fin[10],fout[10];
    int ch,n,i;
    char str[10],name[10][10];
    clrscr();
    for(;;)
    {
        cout<<"1:Std Input to Std Output    2:File to Std Output    3:File to File\n";
        cout<<"Enter your Choice : ";
        cin>>ch;
        switch(ch)
        {
            case 1: cout<<"Enter Number of Records : ";
                    cin>>n;
                    cout<<"Enter "<<n<<" Names : ";
                    for(i=0;i<n;i++)
                    {
                        cin>>name[i];
                    }
                    cout<<"Reversed names are ... "<<endl;
                    for(i=0;i<n;i++)
                    {
                        strrev(name[i]);
                        cout<<name[i];
                        cout<<"\n";
                    }
                    break;
        }
    }
}
```

```

        case 2: cout<<"Enter the Input Filename : ";
                cin>>fin;
                fp1.open(fin,ios::in);
                while(!fp1.fail())
                {
                    fp1>>str;
                    strrev(str);
                    cout<<str;
                    cout<<"\n";
                }
                fp1.close();
                break;
        case 3: cout<<"Enter the Input Filename : ";
                cin>>fin;
                cout<<"Enter the Output Filename : ";
                cin>>fout;
                fp1.open(fin,ios::in);
                fp2.open(fout,ios::out);
                while(!fp1.fail())
                {
                    fp1>>str;
                    strrev(str);
                    fp2<<str;
                    fp2<<"\n";
                }
                fp1.close();
                fp2.close();
                break;

        default: exit(0);
    }
}

```

Working:

Files are objects of the fstream class, so we first declare 2 file objects, one to be associated for input and one to be associated for output. We use strrev() defined in string.h to reverse a string pointed to by str.

Case 1: A set of names is read from the user via cin command, and these are reversed and displayed onto the screen. File I/O is not used here.

Case 2: We create a file with a set of names in it such that each name is in a separate line. We open the file in read mode using open() function of fstream class. We then check whether this open operation actually worked by checking whether file.fail() returned 1 or 0. If it returned 1, then the file could not be opened. This may be because the filename is given incorrectly or maybe because we do not have read permission for that file. Otherwise, we go ahead to access each line (hence each name) in the file using the file>>str statement until we reach the end of

file. file.fail() function also checks for the end of file. It returns 1 if end of file is reached. Once we get each name into an array of characters str, we again use strrev(str) to reverse this and display it on to the screen. After all file operations, we close the file with file.close().

Case 3: The steps in Case 2 are repeated but instead of sending the reversed strings to the screen, we send it to another file. But this file must first be opened in write mode with open() and the flag ios::out .

Output of Program 1:

Execution for option 1: standard input to output

```
1:Std Input to Std Output      2:File to Std Output      3:File to File
Enter your Choice : 1
Enter Number of Records : 3
Enter 3 Names : ramesh
suresh
naresh
Reversed names are ...
hsemar
hserus
hseran
```

Execution for option 2: file to standard output

Before executing create a file in DOS SHELL using command

C:>tc>edit file.txt

Enter some names say

abc

xyz

```
1:Std Input to Std Output      2:File to Std Output      3:File to File
Enter your Choice : 2
Enter the Input Filename : file.txt
cba
zyx
```

Execution for option 3: file to file

```
1:Std Input to Std Output      2:File to Std Output      3:File to File
Enter your Choice : 3
Enter the Input Filename : file1.txt
Enter the Output Filename : file2.txt
```

To verify the output open file2.txt, names should be in reverse order.

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

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<string.h>
#include<process.h>

fstream fp;
int n=0;
char fname[10];

class student
{
    private:
        char usn[10], name[10], branch[5];
    public:
        void pack();
        void unpack(char[]);
        void display();
        void search();
};

void student ::pack()
{
    char buffer[28];
    cout<<"enter the USN,Name and Branch: ";
    cin>> usn>> name>>branch;
    strcpy(buffer,usn);
    strcat(buffer,"|");
    strcat(buffer,name);
    strcat(buffer,"|");
    strcat(buffer,branch);
    strcat(buffer,"|");
    int len=strlen(buffer);
    while(len < 27 )
    {
        strcat(buffer,"#");
        len++;
    }
    buffer[27]='\0';
```

```
    fp<<buffer<<'\n';  
}
```

```
void student::unpack(char buffer[])
```

```
{  
    char *t;  
    t= strtok(buffer," |");  
    cout<<"USN:"<<t<<endl;  
    t=strtok(NULL," |");  
    cout<<"Name:"<<t<<endl;  
    t=strtok(NULL," |");  
    cout<<"Branch:"<<t<<endl;  
}
```

```
void student::display()
```

```
{  
    char buffer[28];  
    int i,j;  
    if(n==0)  
    {  
        cout<<"No records to display";  
        return;  
    }  
    cout<<"\nFile Contents are \n";  
    fp.open(fname,ios::in);  
  
    for(i=1;i<=n;i++)  
    {  
        fp>>buffer;  
        unpack(buffer);  
        cout<<endl;  
    }  
    fp.close();  
}
```

```
void student::search()
```

```
{  
    char buffer[28],temp[28];  
    char key[15];  
    char *usn;  
    int i,j,k,choice;  
    cout<<"Enter the USN to search: ";  
    cin>>key;
```

```
fp.open(fname,ios::in | ios::out);
for(i=1;i<=n;i++)
{
    fp>>buffer;
    strcpy(temp,buffer);
    usn= strtok(temp," | ");
    if(strcmp(key,usn)==0)
    {
        cout<<"\nRecord Found..\n";
        unpack(buffer);
        cout<<endl;
        cout<<"\nDo you wish to modify?\n";
        cout<<"\nEnter your choice \n 1.Yes \n 2.No\n";
        cin>>choice;

        if(choice==1)
        {
            fp.seekp(-27,ios::cur);
            pack();
        }
        fp.close();
        return;
    }
}
cout<<"Record not found..!\n";
fp.close();
getch();
}

void main()
{
    int i,ch;
    student s1;
    clrscr();

    cout<<"Enter the File Name : ";
    cin>>fname;
    fp.open(fname,ios::out | ios::trunc);
    fp.close();
    for(;;)
    {
        cout<<"\n 1.Insert\n 2.Display\n 3.Search and Modify\n 4.Exit \n";
        cout<<"Enter Your Choice\n";
        cin>>ch;
```

```

switch(ch)
{
    case 1:
        n++;
        fp.open(fname,ios::app);
        s1.pack();
        fp.close();
        break;

    case 2:
        s1.display();
        break;

    case 3:
        s1.search();
        break;

    case 4:
        exit(0);

    default:
        cout<<"Invalid Choice..!\n";
}
}
}

```

Working

We are supposed to store information of many students in a file on the hard disk. For each student, we get (from the user) and store his name, USN number and branch. Once we create such a data file, we are supposed to display its contents and be able to search in the file for a particular student's information based on his USN number.

In this program, we are required to store this information in some fixed amount of space, say in x bytes. When data is stored in this way, the data corresponding to each student is called a "fixed-length record". Since each character occupies 1 byte of space, we can store x characters in x bytes. So as to be able to differentiate each of the fields, we store an inter-field separator, which is specified in the question to be the | character. To differentiate between 2 records, we have to store an inter-record separator, and we have chosen the \n character to do this job. Hence, the structure of each fixed-length record that we have defined is as follows:

10 chars		10 chars		5 chars	\n
Name		USN		Dept.	

In our implementation above, a total of 28 spaces or characters are allocated to store a record in the file. The first 10 spaces are for storing the name and the next 10 spaces are for storing the USN number of the student and 5 spaces for storing department. We are going to frequently open and close files, so we do not want to pass around the file object as a parameter to the various functions. Therefore we make the fstream object file as a global parameter. We

then define a class called student, into which we put the different fields in a record as private members, and the functions that work on these fields as public member functions. In pack(), we take an empty buffer of size 28 characters, populate it with a record and then write this buffer to the file which we opened in main(). We read student usn, name, branch from the user. Then using strcpy(buffer,usn), we copy usn entered by the user to the buffer and concatenate the delimiter “|” to the buffer. Similarly it is repeated for name and branch fields. We check length of the buffer, if the buffer length is less than 27 then # is concatenated to the buffer.

Example: buffer contents after copying usn, name and branch fields with delimiter

a	b	c		1	J	B	0	1		I	S	E		#	#	#	#	#	#	#	#	#	#	#	#	#	#	/
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	0

In buffer[27] position ‘/0’ end of the string null character is initialized. Then buffer contents is written to the data file.

In unpack () function, buffer is used as a input parameter to the function. Buffer will contain the record data in which each field needs to be extracted. Each field is separated by “|” delimiter. So to extract strtok function is used in the program. Strtok takes twp parameters, string as first parameter and delimiter “|” as a second parameter a token for separating the string.

```
char *t;
t= strtok(buffer,"|"); // splits the string into tokens ,
cout<<"USN:"<<t<<endl;
//For each subsequent call to get more tokens out of the string, you pass NULL.
t=strtok(NULL,"|");
cout<<"Name:"<<t<<endl;
t=strtok(NULL,"|");
```

The function of display() is implement to display file contents on output screen, opens the data file in read mode using fp1.open(fname,ios::in); for reading, gets each record though the statement file >> buffer and then stores in buffer, unpack(buffer) function is called for the actual unpacking. In unpack(), we use strtok() repeatedly to retrieve all fields between the inter-field separator characters | .

In search(), each record is retrieved from the data file and checked whether this is the USN we are searching for. If it is not, then “record not found” is displayed. If it is, then we say that search is a success, printout the contents of this record using unpack(), If the user decides to modify this found record then,

```
fp.seekp(-27,ios::cur); // used to seeks fp object 27 positions back to previous record
```

pack() is called to read new record data. This is because we want to write the modified record directly to the data file. Here sequential search is used.

Output of the Program

```

Enter the File Name : file.txt

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
2
No records to display
1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
1
Enter the student USN, Name and Branch:
1jb01
abc
ise
1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
1
Enter the student USN, Name and Branch:
1jb02
mno
ise

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
2
File Contents are

USN:1jb01
Name:abc
Branch:ise

USN:1jb02
Name:mno
Branch:ise

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
3
Enter the USN to search: 1jb02

```

```

Record Found..
USN:1jb02
Name:mno
Branch:ise

Do you wish to modify?

Enter your choice
1.Yes
2.No
1
Enter the student USN, Name and Branch:1jb03
jkl
mech
1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
2
File Contents are

USN:1jb01
Name:abc
Branch:ise

USN:1jb03
Name:jkl
Branch:mec

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
3
Enter the USN to search : 1jb04
Record not found

```

File contents before modification:

```

1jb01 | abc | ise#####
1jb02 | mno | ise#####

```

File contents after modification:

```

1jb01 | abc | ise#####

```

1jb03 | jkl | mech#####

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

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
#include<fstream.h>
#include<string.h>

fstream fp1,fp2;
int n=0;
char fname[10];

class student
{
private:
    char usn[15], name[15], branch[5];
public:
    void pack(int);
    void unpack(char[]);
    void display();
    void search();
};

void student ::pack(int a)
{
    char buffer[100];
    cout<<"Enter the student USN, Name and Branch:";
    cin>>usn>>name>>branch;
    strcpy(buffer,usn);
    strcat(buffer," | ");
    strcat(buffer,name);
    strcat(buffer," | ");
    strcat(buffer,branch);
    if(a==1)
    {
        fp1<<buffer;
        fp1<<"\n";
    }
    else
```

```
{
    fp2<<buffer;
    fp2<<"\n";
}
}
void student::display()
{
    char buffer[100];
    int i,j;
    cout<<"\nFile Contents are \n";
    fp1.open(fname,ios::in);
    for(i=1;i<=n;i++)
    {
        fp1>>buffer;
        unpack(buffer);
    }
    fp1.close();
}

void student::unpack(char buffer[])
{
    char *t;
    t = strtok(buffer," | ");
    cout<<"\nUSN : "<<t<<endl;
    t = strtok(NULL," | ");
    cout<<"Name : "<<t<<endl;
    t = strtok(NULL,'\0');
    cout<<"Branch : "<<t<<endl;
}

void student::search()
{
    char buffer[100],temp[100];
    char *usn;
    char key[15];
    int i,choice;
    cout<<"\nEnter the USN to search : ";
    cin>>key;
    fp1.open(fname,ios::in);
    fp2.open("out.txt",ios::out);
    for(i=1;i<=n;i++)
    {
        fp1>>buffer;
        strcpy(temp,buffer);
        usn = strtok(buffer," | ");
```



```
if(strcmp(usn,key)==0)
{
    cout<<"Record Found\n";
    unpack(buffer);
    cout<<"\n Do you wish to modify?"<<endl;
    cout<<"\n1.Yes \n2.No : ";
    cin>>choice;
    if(choice==1)
    {
        pack(2);
        while(!fp1.fail())
        {
            fp1>>buffer;
            fp2<<buffer<<"\n";
        }
        fp1.close();
        fp2.close();
        remove(fname);
        rename("out.txt",fname);
        return;
    }
    else
    {
        fp1.close();
        fp2.close();
        remove("out.txt");
    }
}
else
{
    fp2<<buffer;
    fp2<<"\n";
}
}
cout<<"Record not found";
fp1.close();
fp2.close();
remove("out.txt");
}

void main()
{
    int i,ch;
    student s1;
    clrscr();
```

```

cout<<"Enter the file name : ";
cin>>fname;
fp1.open(fname,ios::out | ios::trunc);
fp1.close();
for(;;)
{
    cout<<"\n 1.Insert\n 2.Display\n 3.Search and Modify\n 4.Exit \n";
    cout<<"Enter Your Choice\n";
    cin>>ch;
    switch(ch)
    {
        case 1:
            n++;
            fp1.open(fname,ios::app);
            s1.pack(1);
            fp1.close();
            break;
        case 2:
            s1.display();
            break;
        case 3:
            s1.search();
            break;
        case 4:
            exit(0);
        default:
            cout<<"Invalid Choice\n";
    }
}
}

```

Working

We are supposed to store information of many students in a file on the hard disk. For each student, we get (from the user) and store his name, USN and branch. Once we create such a data file, we are supposed to display its contents and be able to search in the file for a particular student's information based on his USN number. In this program, we are required to implement variable length records structure to store students' data in to file on the hard disk. Since each character occupies 1 byte of space, we can store x characters in x bytes. So as to be able to differentiate each of the fields, we store an inter-field separator, which is specified in the question to be the | character. To differentiate between 2 records, we have to store an inter-record separator, and we have chosen the \n character to do this job. Hence, the structure of each variable-length record that we have defined is as follows:

15 chars		15 chars		5 chars	\n
----------	--	----------	--	---------	----

USN		Name		Dept.	
-----	--	------	--	-------	--

In the implementation above, file name say file1.txt is read from the user, already file1.txt may contain some data so `fp1.open(fname,ios::out|ios::trunc); fp1.close();` used to clear the file existing contents. To store record into file variable 'n' is used keep track of how many records are stored in file. `fp1.open(fname,ios::app);` is used to open the file in write and append mode. Once the file is opened in append mode, then we can write data into the file, `pack ()` is used to write students data into the file. In `pack()` with parameter equal to 1 is used write students data into the file, since we need to use variable length record structure buffer character array of size 100 bytes is taken to place fields and field delimiters as we read students data from the user. Using `strcpy` function USN is placed in buffer, then `strcat` function is used to append '|' delimiter. Then similarly name and department is placed in to the buffer by separating with '|' delimiter using `strcat` function. After filling the buffer with student USN, name, department with '|' delimiters buffer is written into the file. Then '\n' is written into file to end the record. Similarly `pack ()` is called to add each student records.

The function of `display()` is implement to display file contents on output screen, opens the data file in read mode using `fp1.open(fname,ios::in);` for reading, gets each record though the statement `file >> buffer` and then stores in buffer, `unpack(buffer)` function is called for the actual unpacking. In `unpack()`, we use `strtok()` repeatedly to retrieve all fields between the inter-field separator characters '|'. Control comes back to `main()` and `modify()` is called.

The `search()` is implemented to search and modify based on USN, if user choice is to modify then only particular record is modified otherwise only search and display record if found else display record not found: We open the data file say file1.txt in read mode and a temporary file out.txt in write mode. We search the data file sequentially for the search USN. Let us say that the record with this USN is the 5th record in the file. As we are walking through the records 1-4, we copy these into the out.txt one by one. When the search comes to the 5th record, it finds the USN, prints the respective record's details and asks the user whether he wants to modify it. If the user says yes, then `pack()` is called to get the new details of that student, then the new buffer is written directly to the out.txt. The remaining lines in the file1.txt are then copied to out.txt and then both of them are closed. The outdated file1.txt is deleted and the out.txt is renamed as file1.txt.

In `search()`, each record is retrieved from the data file and checked whether this is the USN we are searching for. If it is not, then this record is copied into the temporary file. If it is, then we say that search is a success, printout the contents of this record using `unpack()`, If the user decides to modify this found record, `pack()` is called with parameter 2. This is because we want to write the modified record directly to the temporary file. `pack()` decides whether to write a buffer to the data file or temporary file depending on the parameter sent to it.

Output of the Program

```
Enter the File Name : file.txt

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
2
No records to display
1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
1
Enter the student USN, Name and Branch:
1jb01
abc
ise
1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
1
Enter the student USN, Name and Branch:
1jb02
mno
ise
```

```
1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
2

File Contents are

USN:1jb01
Name:abc
Branch:ise

USN:1jb02
Name:mno
Branch:ise

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
3
Enter the USN to search: 1jb02
```

```

Record Found..
USN:1jb02
Name:mno
Branch:ise
Do you wish to modify?
Enter your choice
  1.Yes
  2.No
1
Enter the student USN, Name and Branch:1jb03
jkl
mech
  1.Insert
  2.Display
  3.Search and Modify
  4.Exit
Enter Your Choice
2
File Contents are
USN:1jb01
Name:abc
Branch:ise
USN:1jb03
Name:jkl
Branch:mech

```

```

1.Insert
2.Display
3.Search and Modify
4.Exit
Enter Your Choice
3
Enter the USN to search : 1jb04
Record not found

```

File contents before modification:

```

1jb01 | abc | ise
1jb02 | mno | ise

```

File contents before modification:

```

1jb01 | abc | ise
1jb03 | jkl | mech

```

4. Write a C++ program to read and write student objects with variable-length records using any suitable record structure and to read from this file a student record using RRN.

```

#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<values.h>
#include<stdio.h>
#include<fstream.h>
#include<string.h>
fstream fp;
int n=0;
char fname[10],index[10];

class student
{
private:
    char name[15],usn[15],branch[5];
public:
    void pack();
    void unpack(char[]);
    void display();
    void searchbyrrn();

```

```
};

void student::pack()
{
    char buffer[100];
    cout<<"Enter the student USN, Name and Branch:";
    cin>>usn>>name>>branch;
    strcpy(buffer,usn);
    strcat(buffer," | ");
    strcat(buffer,name);
    strcat(buffer," | ");
    strcat(buffer,branch);
    index[n-1] = fp.tellg();
    fp<<buffer;
    fp<<"\n";
}

void student::unpack(char buffer[])
{
    char *t;
    t = strtok(buffer," | ");
    cout<<"\nUSN : "<<t<<endl;
    t = strtok(NULL," | ");
    cout<<"Name : "<<t<<endl;
    t = strtok(NULL,'\0');
    cout<<"Branch : "<<t<<endl;
}

void student::display()
{
    char buffer[100];
    int i,j;
    cout<<"\nFile Contents are \n";
    fp.open(fname,ios::in);
    for(i=1;i<=n;i++)
    {
        fp>>buffer;
        unpack(buffer);
    }
    fp.close();
}

void student::searchbyrrn()
{
    int rrn;
```

```
char buffer[30];
cout << "\nEnter RRN of the record:";
cin>>rrn;
fp.open(fname,ios::in);
if(rrn > n-1)
{
    cout <<"Record with RRN "<<rrn<<" was not found..\n";
    fp.close();
    return;
}
fp.seekg(index[rrn],ios::beg);
fp>>buffer;
unpack(buffer);
fp.close();
return;
}

void main()
{
    int i,ch;
    student s1;
    clrscr();
    cout<<"\nEnter the Filename:";
    cin>>fname;
    fp.open(fname,ios::out | ios::trunc);
    fp.close();
    for(;;)
    {
        cout<<"\n 1.Insert \n 2.Display \n 3.Search By RRN \n 4.Exit \n";
        cout<<"Enter your Choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:
                n++;
                fp.open(fname,ios::out | ios::ate);
                s1.pack();
                fp.close();
                break;
            case 2:
                s1.display();
                break;
            case 3:
                s1.searchbyrrn();
                break;
```

```
        default:  
        exit(0);  
    }  
}  
}
```

Working:

RRN is a direct access method to access a record that emerges from viewing a file as a collection of records rather than as a collection of bytes. If a file is a sequence of records, the RRN of a record gives its position relative to the beginning of the file. The first record in a file has RRN 0, the next has RRN 1 and so forth.

In general, given a fixed length record file where the record size is r , the byte offset of record with a RRN of n is

$$\text{Byte offset} = n * r$$

The above program is to store and search records of variable length record, using RRN direct access method. In the implementation above, file name say file1.txt is read from the user, already file1.txt may contain some data so `fp1.open(fname,ios::out|ios::trunc); fp1.close();` used to clear the file existing contents. To store record into file variable 'n' is used keep track of how many records are stored in file. `fp1.open(fname,ios::app);` is used to open the file in write and append mode. Once the file is opened in append mode, then we can write data into the file, `pack()` is used to write students data into the file. In `pack()` is used to write students data into the file, since we need to use variable length record structure buffer character array of size 100 bytes is taken to place fields and field delimiters as we read students data from the user. Using `strcpy` function USN is placed in buffer, then `strcat` function is used to append '|' delimiter. Then similarly name and department is placed in to the buffer by separating with '|' delimiter using `strcat` function. After filling the buffer with student USN, name, department with '|' delimiters buffer is written into the file. Then '\n' is written into file to end the record. Similarly `pack()` is called to add each student records. index array is used to store the length of each records `fp.tellg()` is used to access the offset of the record. In function `search()`, user enters rrn and checks if $rrn > n-1$ (number of records), then display "error message rrn is not found" otherwise using `fp.seekg(index[rrn],ios::beg);` function it seeks fp directly to the beginning of the required record.

Output of the Program


```

Enter the Filename:file.txt

1.Insert
2.Display
3.Search By RRN
4.Exit
Enter your Choice:1
Enter the student USN, Name and Branch:1
abc
ise

1.Insert
2.Display
3.Search By RRN
4.Exit
Enter your Choice:1
Enter the student USN, Name and Branch:2
mno
ise
1.Insert
2.Display
3.Search By RRN
4.Exit
Enter your Choice:2

File Contents are

USN : 1
Name : abc
Branch :ise

USN : 2
Name : mno
Branch :ise

1.Insert
2.Display
3.Search By RRN
4.Exit
Enter your Choice:3

```

```

Enter RRN of the record:2
Record with RRN 2 was not found..

1.Insert
2.Display
3.Search By RRN
4.Exit
Enter your Choice:3

Enter RRN of the record:0

USN : 1
Name : abc
Branch :ise

```

File Contents:

```

1 | abc | ise
2 | mno | ise

```

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

```

#include<iostream.h>
#include<process.h>
#include<conio.h>
#include<fstream.h>
#include<string.h>
#include<stdio.h>
fstream fp;

```

```
int recount=0;
char fname[10];

class student
{
private:
    char usn[15];
    char name[15];
    char branch[5];
public:
    void pack(char[]);
    void unpack(char[]);
    void display();
    void insrec();
    void delrec();
    int search(char[]);
};

struct memindex
{
    char key[15];
    int offset;
};

memindex index[10];

void student ::pack(char buffer[])
{
    cout<<"Enter the student USN, Name and Branch:";
    cin>>usn>>name>>branch;
    strcpy(buffer,usn);
    strcat(buffer," | ");
    strcat(buffer,name);
    strcat(buffer," | ");
    strcat(buffer,branch);
}

void student::insrec()
{
    char buffer[50],tem[20];
    char *usn;
    int i,j,pos;
    fp.open(fname,ios::out | ios::ate);
    pack(buffer);
    strcpy(tem,buffer);
```

```
    usn=strtok(tem," | ");
    pos=search(usn);
    if(pos!=0)
    {
        cout<<"USN already Exists \n"<<endl;
        fp.close();
        return;
    }
    recount++;
    strcpy(index[recount].key,usn);
    index[recount].offset=fp.tellg();
    fp<<buffer;
    fp<<"\n";
    memindex temp;

    for(i=1;i<recount;i++)
    {
        for(j=1;j<recount;j++)
        {
            if(strcmp(index[j].key,index[j+1].key)>0)
            {
                strcpy(temp.key,index[j].key);
                temp.offset=index[j].offset;

                strcpy(index[j].key,index[j+1].key);
                index[j].offset=index[j+1].offset;

                strcpy(index[j+1].key,temp.key);
                index[j+1].offset=temp.offset;
            }
        }
    }
    fp.close();
}

void student::display()
{
    char buffer[50];
    int i,j;
    if(recount == 0)
    {
        cout<<"No Records to Display"<<endl;
        return;
    }
    cout<<"\t-----"<<endl;
```

```

cout<<"\t Primary index"<<endl;
cout<<"\t-----"<<endl;
cout<<"\t key\t"<<"offset"<<endl;
cout<<"\t-----"<<endl;
for(i=1;i<=recount;i++)
{
    cout<<"\t"<<index[i].key<<"\t"<<index[i].offset<<endl;
}
cout<<"File Contents are"<<endl;
fp.open(fname,ios::in);
for(i=1;i<=recount;i++)
{
    fp.seekg(index[i].offset,ios::beg);
    fp>>buffer;
    unpack(buffer);
    cout<<endl;
}
fp.close();
}

```

void student::delrec()

```

{
    char usn[15];
    int pos;
    fp.open(fname,ios::out | ios::ate);
    cout<<"\nEnter the USN:";
    cin>>usn;
    pos=search(usn);
    if(pos==0)
    {
        cout<<"\nRecord not found";
        return;
    }
    fp.seekg(index[pos].offset,ios::beg);
    fp<<"*";
    fp.close();
    for(int i=pos;i<recount;i++)
    {
        strcpy(index[i].key,index[i+1].key);
        index[i].offset=index[i+1].offset;
    }
    cout<<"Record Deleted Successfully"<<endl;
    recount--;
}

```

```
int student::search(char usn[])
{
    int low=1,high=recount,mid;
    while(low<=high)
    {
        mid=(low+high)/2;
        if(strcmp(index[mid].key,usn)==0)
            return mid;
        if(strcmp(index[mid].key,usn)>0)
            high=mid-1;
        else
            low=mid+1;
    }
    return 0;
}

void student::unpack(char buffer[])
{
    char *t;
    t = strtok(buffer," | ");
    cout<<"USN : "<<t<<endl;
    t = strtok(NULL," | ");
    cout<<"Name : "<<t<<endl;
    t = strtok(NULL,'\0');
    cout<<"Branch : "<<t<<endl;
}

void main()
{
    char buffer[50];
    student s1;
    char usn[15];
    int i,j,k,choice,pos;
    clrscr();
    cout<<"Enter the file name: ";
    cin>>fname;
    fp.open(fname,ios::out | ios::trunc);
    fp.close();
    for(;;)
    {
        cout<<endl;
        cout<<" 1.Insert\n 2.Delete\n 3.Display\n 4.Search\n 5.Exit\n";
        cout<<"Enter your choice\n";
        cin>>choice;
        switch(choice)
```

```

{
    case 1:
        s1.insrec();
        break;
    case 2:
        s1.delrec();
        break;
    case 3:
        s1.display();
        break;
    case 4:
        cout<<"\nEnter the USN to be searched:";
        cin>>usn;
        pos=s1.search(usn);
        if(pos==0)
        {
            cout<<"Record not found..\n";
            break;
        }
        fp.open(fname,ios::in);
        fp.seekg(index[pos].offset,ios::beg);
        fp>>buffer;
        s1.unpack(buffer);
        fp.close();
        break;
    default:
        exit(0);
}
}
}

```

Working:

An index is a tool for finding records in a file. It consists of a key field on which the index is searched and reference (offset) field that tells where to find the data file record associated with a particular key.

Index		Record file	
Key	Reference field	Address of record	Actual data record
1jb01	29	0	1jb02 abc ise
1jb02	0	15	1jb03 xyz ise
1jb03	15	29	1jb01 pqr ise

1jb04	57
1jb05	43

43	1jb05 mno ise
57	1jb04 asd ise

In C++ program 5 implementation USN-based index (primary key) of the file in main memory is implemented, where sorting, and hence searching, is faster. Such an index is referred to as the primary index, because the USN is the primary key of the student database. Once the index is sorted on USN, binary search can be used for searching. When new records are inserted into the file, an appropriate entry is also added to the index in a way so as to maintain the sorted order of the index. When a record is deleted from the file, the corresponding entry in the index is also deleted and all the index entries below the deleted entry are pushed up by one (still maintaining sorted order). We implement this technique in this program. We first create a structure called memindex, which contains a character field of 15 spaces to store a primary key (USN, in our case) and an integer field to store the byte offset of each record in the file. When we access an index entry, we get the byte offset of the record to which this index is pointing to, and we use seekg() to jump to the record directly without walking through the file in a serial manner. We create 10 elements of this list here, but if more than 10 records are to be indexed, then this number has to be increased. To know how many index elements are present, we use a global variable called recount and initialize it to 0 as the index is empty to start with.

In main(), we open a data file for writing and call insrec() only once for one student's information, along with the buffer in which it has to pack the information. The pack() gets the data of this student from the user and packs it using the | as the inter-field separator and the '\n' as the inter-record separator. This packed buffer is then written to the data file. At this point in the program, we know that there is only one record in the data file, since we have allowed only one record to go to the file yet. Then populate the first primary index entry with the USN and byte offset. This being the very first record, its byte offset is 0. We increment the global variable recount by 1 because the index now has exactly one entry.

Now that one record is placed into the file and a corresponding entry is put into the index, In insrec(), we open the data file in ate mode (append-to-end), so that whatever record we write to the file gets added to the end of the file after setting the put pointer to the end. We call pack() to pack a new buffer for us. We extract the USN field from buffer using strtok to search the index to see whether the USN that is in this buffer already exists. If it does, we do not insert the current record into the file and return to main(). If not, we continue with insertion.

Since recount gives the total number of index entries, it is incremented by 1 to make space for the new entry, and the USN and byte offset of the record to be inserted are written into the newly created position in the index. The byte offset of the record being inserted is given by tellg() which returns a long integer, so we have to typecast it to make it a normal integer. After adding data into file each time index is sorted based USN using **Bubble sorting technique**. To delete a record from the file, from the main() , delrec() is called, before deleting a record file is opened in ate mode. User enter the USN of the record to be deleted, then search () is called with the parameter USN, we use **Binary searching technique** search the index to see whether such a USN actually exists or not. If USN does not exists, search () returns 0 otherwise the position of the USN found in index. If pos is 0 display "Record not found" and return to main (). If pos is non-zero, we move the fp object to point to the particular pos offset using fp.seekg(index[pos].offset,ios::beg); . Then "*" is placed to mark that record is deleted. During

deletion, we do not actually delete anything from the data file. We just delete the index entry for that particular record. The index entry for the record we want to delete is found in position i in the index. We pull up all entries from $i+1$ position by 1. Finally recount is decremented by 1.

Output of the program

```
Enter the file name: file.txt
```

```

1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice
3
No Records to Display

1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice
1
Enter the student USN, Name and Branch:1
abc
ise
1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice
1
Enter the student USN, Name and Branch:2
def
civ

1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice
1
Enter the student USN, Name and Branch:0
mno
mec_
1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice
```

```
3
```

```
-----
Primary index
-----
```

```
key    offset
```

```
-----
```

```
0      22
```

```
1      0
```

```
2      11
```

```
File Contents are
```

```
USN : 0
```

```
Name : mno
```

```
Dept :mec
```

```
USN : 1
```

```
Name : abc
```

```
Dept :ise
```

```
USN : 2
```

```
Name : def
```

```
Dept :civ
```

```
1.Insert
```

```
2.Delete
```

```
3.Display
```

```
4.Search
```

```
5.Exit
```

```
enter your choice
```

```
4
```

```
Enter the USN to be searched:3
```

```
Record not found
```

```
1.Insert
```

```
2.Delete
```

```
3.Display
```

```
4.Search
```

```
5.Exit
```

```
enter your choice
```



```

4
Enter the USN to be searched:2
USN : 2
Name : def
Dept :civ

```

```

1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice

```

```

2
Enter the USN:1
Record Deleted Successfully

```

```

1.Insert
2.Delete
3.Display
4.Search
5.Exit
enter your choice

```

```

3
-----
Primary index
-----
key    offset
-----
0      22
2      11

```

```

File Contents are
USN : 0
Name : mno
Dept :mec

```

```

USN : 2
Name : def
Dept :civ

```

File Contents after Deletion:

```

1|abc|ise
2|def|civ
0|mno|mec

```

File Contents after Deletion:

```

*|abc|ise
2|def|civ
0|mno|mec

```

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

```
#include<iostream.h>
#include<process.h>
#include<conio.h>
#include<fstream.h>
#include<string.h>
#include<stdio.h>

fstream fp;
int recount=0;
char fname[10];

class student
{
private:
    char name[15];
    char usn[15];
    char branch[5];
public:
    void pack(char[]);
    void unpack(char[]);
    void display();
    void insrec();
    void delrec();
    int search(char[]);
    void sch_scn_index();
};

struct primaryindex
{
    char pkey[15];
    int offset;
};

struct secondaryindex
{
    char skey[15];
    char pkey[15];
};

student s1;
```

```
primaryindex pindex[20];
secondaryindex sindex[20];
```

```
void student::pack(char buffer[])
```

```
{
    cout<<"Enter the student USN, Name and Branch:";
    cin>>usn>>name>>branch;
    strcpy(buffer,usn);
    strcat(buffer," | ");
    strcat(buffer,name);
    strcat(buffer," | ");
    strcat(buffer,branch);
}
```

```
void student::insrec()
```

```
{
    char buffer[50],tem[20];
    char *usn,*name;
    int i,j,pos;
    fp.open(fname,ios::out | ios::ate);
    pack(buffer);
    strcpy(tem,buffer);
    usn=strtok(tem," | ");
    name=strtok(NULL," | ");
    pos=search(usn);
    if(pos!=0)
    {
        cout<<"USN already present\n";
        fp.close();
        return;
    }
    recount++;
    strcpy(pindex[recount].pkey,usn);
    pindex[recount].offset=fp.tellg();

    strcpy(sindex[recount].skey,name);
    strcpy(sindex[recount].pkey,usn);

    fp<<buffer;
    fp<<"\n";
    primaryindex temp;
```

```

for(i=1;i<recount;i++)
{
    for(j=1;j<recount;j++)
    {
        if(strcmp(pindex[j].pkey,pindex[j+1].pkey)>0)
        {
            strcpy(temp.pkey,pindex[j].pkey);
            temp.offset=pindex[j].offset;
            strcpy(pindex[j].pkey,pindex[j+1].pkey);
            pindex[j].offset=pindex[j+1].offset;

            strcpy(pindex[j+1].pkey,temp.pkey);
            pindex[j+1].offset=temp.offset;
        }
    }
}
fp.close();
}

```

void student::display()

```

{
    char buffer[50];
    int i,j;
    fp.open(fname,ios::in);
    if(recount == 0)
    {
        cout<<"No Records to Display"<<endl;
        fp.close();
        return;
    }
    cout<<"\t-----"<<endl;
    cout<<"\t Primary index"<<endl;
    cout<<"\t-----"<<endl;
    cout<<"\t key  "<<"offset"<<endl;
    cout<<"\t-----"<<endl;
    for(i=1;i<=recount;i++)
    {
        cout<<"\t"<<pindex[i].pkey<<"\t"<<pindex[i].offset<<endl;
    }

    cout<<"\t-----"<<endl;
    cout<<"\t  Secondary index"<<endl;
    cout<<"\t-----"<<endl;
}

```

```

    cout<<"\t sec-key\t"<<"pri-key"<<endl;
    cout<<"\t-----"<<endl;
    for(i=1;i<=recount;i++)
    {
        cout<<"\t"<<sindex[i].skey<<"\t\t"<<sindex[i].pkey<<endl;
    }
    cout<<"\t-----"<<endl;
    cout<<"File Contents are"<<endl;
    for(i=1;i<=recount;i++)
    {
        fp.seekg(pindex[i].offset,ios::beg);
        fp>>buffer;
        unpack(buffer);
    }
    fp.close();
}

```

void student::delrec()

```

{
    char usn[15];
    int pos,j;
    cout<<"\nEnter the usn:";
    cin>>usn;
    fp.open(fname,ios::out | ios::ate);
    pos=search(usn);
    if(pos==0)
    {
        cout<<"\n Record not found";
        fp.close();
        return;
    }
    fp.seekg(pindex[pos].offset,ios::beg);
    fp<<"*";
    fp.close();
    for(int i=pos;i<recount;i++)
    {
        strcpy(pindex[i].pkey,pindex[i+1].pkey);
        pindex[i].offset=pindex[i+1].offset;
    }
    j=1;
    while(strcmp(sindex[j].pkey,usn)!=0)
    {
        j++;
    }
}

```

```
    }
    for(i=j;i<recount;i++)
    {
        strcpy(sindex[i].skey,sindex[i+1].skey);
        strcpy(sindex[i].pkey,sindex[i+1].pkey);
    }
    cout<<"Record deleted successfully"<<endl;
    recount--;
}
```

```
void student::sch_scn_index()
{
    char buffer[50];
    char name[15];
    int i,j;
    int pos,flag=0;
    fp.open(fname,ios::in);
    cout<<"Enter the secondary key(name) \n";
    cin>>name;
    for(i=1;i<=recount;i++)
    {
        if(strcmp(sindex[i].skey,name)==0)
        {
            flag=1;
            pos=search(sindex[i].pkey);
            fp.seekg(pindex[pos].offset,ios::beg);
            fp>>buffer;
            unpack(buffer);
        }
    }
    if(flag==0)
    {
        cout<<"Record not found";
    }
    fp.close();
}
```

```
void student::unpack(char buffer[])
{
    char *t;
    t = strtok(buffer," |");
    cout<<"USN : "<<t<<endl;
    t = strtok(NULL," |");
```

```
    cout<<"Name : "<<t<<endl;
    t = strtok(NULL,'\0');
    cout<<"Branch : "<<t<<endl;
}

int student::search(char usn[])
{
    int low=1,high=recount,mid;
    while(low<=high)
    {
        mid=(low+high)/2;
        if(strcmp(pindex[mid].pkey,usn)==0)
            return mid;
        if(strcmp(pindex[mid].pkey,usn)>0)
            high=mid-1;
        else
            low=mid+1;
    }
    return 0;
}

void main()
{
    char buffer[50];
    char usn[15];
    int i,j,k,choice,pos;
    clrscr();
    cout<<"Enter the file name: ";
    cin>>fname;
    fp.open(fname,ios::out | ios::trunc);
    fp.close();
    for(;;)
    {
        cout<<"\n1.Insert\n2.Delete\n3.Display\n";
        cout<<"4.Search using primary key\n";
        cout<<"5.Search using secondary key\n6.Exit\n";
        cout<<"Enter your choice\n";
        cin>>choice;
        switch(choice)
        {
            case 1:
                s1.insrec();
                break;
            case 2:
```

```
        s1.delrec();
        break;
    case 3:
        s1.display();
        break;
    case 4:
        cout<<"\nEnter the usn to be searched:";
        cin>>usn;
        pos=s1.search(usn);
        if(pos==0)
        {
            cout<<"Record not found\n";
            break;
        }
        fp.open(fname,ios::in);
        fp.seekg(pindex[pos].offset,ios::beg);
        fp>>buffer;
        s1.unpack(buffer);
        fp.close();
        break;
    case 5:
        s1.sch_scn_index();
        break;

    default:
        exit(0);
}
}
```

Working:

Secondary Index: It contains secondary key and primary key as reference. In this program, name is used as the Secondary key and USN is used as the reference for the secondary key.

When a record is inserted in the data file, the USN is extracted from the record and an entry is made in the primary index along with the byte offset of the record. The name is extracted and an entry is made in the secondary index with USN as the reference. Primary index is kept in sorted order by USN.

Deletion of a record results in the deletion of the corresponding entry from both the primary index and the secondary index.

Search by USN: User enters USN to be searched as the key. Binary search is applied on the primary index to search for the USN. If USN is found, we get the byte offset and file stream object moves to the particular position to retrieve the record related of that USN from record file.

Search by name: User enters the Name to be searched. Linear search is applied to the secondary index to search for the name. If the name is found then, the USN for the particular name is retrieved from secondary index using USN byte offset is accessed from primary index, once we get the byte offset then file stream object is moved to the particular position to retrieve the record related of that USN from record file.

Secondary Index		Primary Index		Address of record	Record file
Secondary Key	Primary Key	Primary Key	Reference field		Actual data record
abc	1jb02	1jb01	29	0	1jb02 abc ise
xyz	1jb03	1jb02	0	15	1jb03 xyz ise
pqr	1jb01	1jb03	15	29	1jb01 pqr ise
mno	1jb05	1jb04	57	43	1jb05 mno ise
asd	1jb04	1jb05	43	57	1jb04 asd ise

```

Enter the file name: file.txt

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
1
Enter the student USN, Name and Branch:1jb03
abc
ise

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
1
Enter the student USN, Name and Branch:1jb05
abc
cse

```

```
1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
1
Enter the student USN, Name and Branch:1jb01
jkl
ise

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
3
-----
Primary index
-----
key  offset
-----
1jb01  30
1jb03  0
1jb05  15
-----

Secondary index
-----
sec-key      pri-key
-----
abc          1jb03
abc          1jb05
jkl          1jb01
-----

File Contents are
USN : 1jb01
Name : jkl
Branch :ise

USN : 1jb03
Name : abc
Branch :ise

USN : 1jb05
Name : abc
Branch :cse
```

```
1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
4

Enter the usn to be searched:1jb03
USN : 1jb03
Name : abc
Branch :ise

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
5_
Enter the secondary key(name)
abc
USN : 1jb03
Name : abc
Branch :ise
USN : 1jb05
Name : abc
Branch :cse

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
2
```

```
Enter the usn:1jb03
Record deleted successfully

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
4

Enter the usn to be searched:1jb03
Record not found

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
5
Enter the secondary key(name)
abc
USN : 1jb05
Name : abc
Branch :cse

1.Insert
2.Delete
3.Display
4.Search using primary key
5.Search using secondary key
6.Exit
Enter your choice
3

-----
Primary index
-----
key  offset
-----
1jb01  30
1jb05  15
-----

Secondary index
-----
sec-key      pri-key
-----
abc          1jb05
jkl          1jb01
-----
```

```
File Contents are
```

```
USN : 1jb01
```

```
Name : jkl
```

```
Branch :ise
```

```
USN : 1jb05
```

```
Name : abc
```

```
Branch :cse
```

File Contents before Deletion

1jb03 | abc | ise

1jb05 | abc | cse

1jb01 | jkl | ise

File Contents after Deletion

*jb03 | abc | ise

1jb05 | abc | cse

1jb01 | jkl | ise

7. Write a C++ 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 lists.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<string.h>

fstream fp1,fp2,fp3;

void match()
{
    char buffer1[100],buffer2[100];
    int i,j;
    fp1.open("names1.txt",ios::in);
    fp2.open("names2.txt",ios::in);
    fp3.open("match.txt",ios::out);

    fp1>>buffer1;
    fp2>>buffer2;

    cout<<"Matched Names are ..... "<<endl;

    while(!fp1.fail() && !fp2.fail())
    {
        if(strcmp(buffer1,buffer2)==0)
        {
            fp3<<buffer1;
            fp3<<"\n";
            cout<<buffer1;
            cout<<"\n";
            fp1>>buffer1;
            fp2>>buffer2;
        }

        else if(strcmp(buffer1,buffer2)<0)
        {
            fp1>>buffer1;
        }

        else
        {
            fp2>>buffer2;
        }
    }
}
```

```

    }
}
}

void main()
{
    clrscr();
    match();
    getch();
}

```

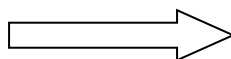
Working:

Consequential operations involve the coordinated processing of two or more sequential lists to produce a single output list.

Matching Names in two lists:

We want to output the names common to the two files Names1.txt and names2.txt shown in table below. This operation is called as match operation or an intersection. Assume that the lists are sorted in ascending order.

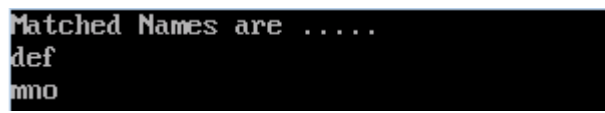
Names1. txt	Names2.txt	Match.txt
Abhilash	Abhilash	Abhilash
Ajay	Amar	Chethan
Chethan	Bharath	Ranjith
Kiran	Chethan	
Ranjith	Mahesh	
Umesh	Ranjith	



We begin by reading the initial names from each files and find they match. We output the first name of names1.txt to match.txt as a name matches. Then we read the next names each file. Now next name in names1.txt is less than names2.txt, we are trying to match name "Ajay" and "Amar", since it is less, we scan down in names1.txt and read next name "chethan". Now we are trying to match "chethan" and "Amar". Name in names1.txt is greater than name in names2.txt. So scan down in names2.txt. read next name that is

“bharath” in names2.txt. Now we are trying to match “chethan” and “bharath”, Name in names1.txt is greater than name in names2.txt. So scan down in names2.txt. read next name that is “chethan” in names2.txt. Now we are trying to match “chethan” and “chethan”, it matches output the name of names1.txt to match.txt. Then we read the next names from each file.

Output of the program



```
Matched Names are .....  
def  
mno
```

Contents of names1.txt

```
abc  
def  
jkl  
mno
```

Contents of names2.txt

```
def  
mno  
xyz
```

Contents of match.txt

```
def  
mno
```


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

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void main()
{
    char name[20][20];
    int i,low;
    fstream fp1;
    fstream fp[10];
    fp[1].open("n1.txt",ios::in);
    fp[2].open("n2.txt",ios::in);
    fp[3].open("n3.txt",ios::in);
    fp[4].open("n4.txt",ios::in);
    fp[5].open("n5.txt",ios::in);
    fp[6].open("n6.txt",ios::in);
    fp[7].open("n7.txt",ios::in);
    fp[8].open("n8.txt",ios::in);

    fp1.open("merge.txt",ios::out);
    for(i=1;i<=8;i++)
    {
        fp[i]>>name[i];
    }

    for(;;)
    {
        low=1;
        for(i=1;i<=8;i++)
        {
            if(strcmp(name[i],name[low])<0)
            {
                low=i;
            }
        }
        if(strcmp(name[low],"~")==0)
        {
            fp1.close();
            for(i=1;i<=8;i++)
```

```

        {
            fp[i].close();
        }
        exit(0);
    }
    else
    {
        fp1<<name[low];
        fp1<<"\n";
        cout<<name[low]<<"\n";
        fp[low]>>name[low];
        if(fp[low].fail())
        {
            strcpy(name[low],"~");
        }
    }
}
}

```

File Contents are

n1.txt	n2.txt	n3.txt	n4.txt	n5.txt	n6.txt	n7.txt	n8.txt
Ball Mango	Apple Orange Rat	Carrot Fish	Girl Hand	Hammer Stone	Sun Top	Ant Hill	Bill Kill

Merge.txt

Ant
 Apple
 Ball
 Bill
 Carrot
 Fish
 Girl
 Hammer
 Hand
 Hill
 Kill
 Mango
 Orange
 Rat
 Stone
 Sun
 Top

File Structures Lab Viva Questions:**1. What are file structures?**

A pattern for arranging data in a file. It is a combination of representations for data in files and of operations for accessing the data.

2. Explain physical files and logical files?

Physical file: A file as seen by the operating system, and which actually exists on secondary storage.

Logical file: A file as seen by a program.

3. Explain open function of fstream class with syntax?

```
fd = open(filename, flags[, pmode]);
```

fd-file descriptor

The following **flags** can be bitwise *ored* together for the **access mode**:

O_RDONLY : Read only

O_WRONLY : Write only

O_RDWR : Read or write

O_CREAT : Create file if it does not exist

O_EXCL : If the file exists, truncate it to a length of zero, destroying its contents.
(used only with O_CREAT)

O_APPEND : Append every write operation to the end of the file

O_TRUNC : Delete any prior file contents

Pmode- protection mode

4. What are the usage of tellg(), tellp(), seekg(), seekp() functions?

streampos tellg(): Get position in input sequence. Returns the position of the current character in the input stream.

streampos tellp(): Get position in output sequence. Returns the position of the current character in the output stream.

These two member functions with no parameters return a value of the member type streampos, which is a type representing the current get position (in the case of tellg) or the put position (in the case of tellp).

seekg() and seekp():

These functions allow to change the location of the get and put positions. Using this prototype, the stream pointer is changed to the absolute position position (counting from the beginning of the file).

```
seekg ( offset, direction );
```

```
seekp ( offset, direction );
```

Using this prototype, the get or put position is set to an offset value relative to some specific point determined by the parameter direction. offset is of type streamoff. And direction is of

type `seekdir`, which is an enumerated type that determines the point from where offset is counted from, and that can take any of the following values:

direction	Description
<code>ios::beg</code>	offset counted from the beginning of the stream
<code>ios::cur</code>	offset counted from the current position
<code>ios::end</code>	offset counted from the end of the stream

5. Explain why buffer is used in read and write operations?

Buffer is used for temporary storage during file read and write operations

6. Explain the different costs of disk access?

access time: The total time required to store or retrieve data.

transfer time: The time required to transfer the data from a sector, once the transfer has begun.

seek time: The time required for the head of a disk drive to be positioned to a designated cylinder.

rotational delay: The time required for a designated sector to rotate to the head of a disk drive

7. Explain the functions read and write with parameters?

Reading

The C++ [read](#) function is used to read data from a file for handle level access.

The read function must be supplied with (as an arguments):

- The source file to read from
- The address of the memory block into which the data will be stored
- The number of bytes to be read(byte count)

The value returned by the *read* function is the number of bytes read.

Read function:

```
Read (Source_file, Destination_addr, Size)
```

Writing

The C++ [write](#) function is used to write data to a file for handle level access.

The write function must be supplied with (as an arguments):

- The logical file name used for sending data
- The address of the memory block from which the data will be written
- The number of bytes to be write

The value returned by the *write* function is the number of bytes written.

Write function:

```
Write(Destination_file, Source_addr, Size)
```

8. Explain I/O redirection?

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")
```

9. What are the different methods of accessing records?

Sequential access: Accessing data from a file whose records are organized on the basis of their successive physical positions.

Direct Access : Accessing data from a file by record position with the file, without accessing intervening records.

10. Distinguish between internal and external fragmentation. Describe the remedial measures to minimize fragmentation?

Internal fragmentation: Fragmentation in which the unused space is within the allocated areas.

External fragmentation: Fragmentation in which the unused space is outside of the allocated areas.

11. Define RRN

Relative record number: An ordinal number indicating the position of a record within a file

12. What are the different placement strategies?

First fit: A placement strategy which selects the first space on the free list which is large enough.

Best fit: A placement strategy which selects the smallest space from the free list which is large enough.

Worst fit: A placement strategy which selects the largest space from the free list (if it is large enough.)

13. What is a record? What are the ways in which records can be organized in a file?

A record is a collection of related data fields which provide information. A subdivision of a file, containing data related to a single entity.

Record Organization:

Make records a predictable number of bytes.

Make records a predictable number of fields.

Begin each record with a length indicator

Use an index to keep track of addresses.

Place a delimiter at the end of each record.

14. What do you understand by index? Explain simple index for sequential files?

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.

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.

15. Explain the key-sorting techniques and their limitations

Keysorting is a way to sort medium size files.

Description of the Method

- Read each record sequentially into memory, one by one
- Save the key of the record, and the location of the record, in an array (*KEYNODES*).
- After all records have been read, internally sort the *KEYNODES* array of record keys and locations.
- Using the *KEYNODES* array, read each record back into memory a second time using direct access.

Limitations of the Keysort Method

- Keysort is only possible when the *KEYNODES* array is small enough to be held in memory.
- Each record must be read twice: once sequentially and once directly.
- The direct reads each require a seek.
- If the original file and the output file are on the same physical drive, there will also be a seek for each write.
- Keysorting is a way to sort medium size files.

16. Explain the concept of B-Trees in multilevel indexing

Tree structured indexes such as B-trees are a scalable alternative to simple indexes. A multiway tree in which all insertions are made at the leaf level. New nodes at the same level are created when required by node overflow, and new nodes at the parent level are created when required by the creation of new nodes.

17. Explain the limitations of binary searching and internal sorting?

- Binary searching requires more than one or two accesses.
- More than one or two accesses is too many.
- Keeping a file sorted is very expensive.
- An internal sort works only on small files.

18. Explain the operations required to maintain the index files?

Create empty index file and data files

Load index file into memory before using it

Rewrite the index file from memory after using it
Add records to data file
Delete records from data file
Update records in data file
Update the index to reflect the changes in the data file

19. Explain co-sequential processing?

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.

20. Explain K-Way merging algorithm?

k-way merge: A merge of order k.

21. Explain redistribution of elements in a B-Tree?

Redistribution: The movement of contents between adjacent nodes to equalize the loading.

22. Explain the following terms: hashing, double hashing?

Hashing: The transformation of a search key into a number by means of mathematical calculations.

Double hashing: A collision resolution scheme which applies a second hash function to keys which collide, to determine a probing distance.

23. Why extendible hashing is required?

An application of hashing that works well with files that over time undergo substantial changes in size.

24. What is bucket?

Bucket: An area of a hash table with a single hash address which has room for more than one record.

25. Explain the different collision resolution techniques?

Progressive overflow: A collision resolution technique which places overflow records at the first empty address after the home address

Storing more than One Record per Address-Buckets: An area of a hash table with a single hash address which has room for more than one record.

Double Hashing: A collision resolution scheme which applies a second hash function to keys which collide, to determine a probing distance.

Open Chaining: Open chaining forms a linked list, or chain, of synonyms.

Scatter Tables: If all records are moved into a separate "overflow" area, with only links being left in the hash table, the result is a *scatter table*.