SRINIVAS INSTITUTE OF TECHNOLOGY

(Accredited by NAAC)

Valachil, Mangaluru-574143

Department of Information Science and Engineering

LAB MANUAL

FILE STRUCTURES LABORATORY WITH MINI PROJECT
18ISL67

6th Semester B.E.

2020-2021

Name:	
USN:	
Section:	
Batch:	

Compiled By: Mrs.Mamatha, Assistant Professor

Srinivas Institute of Technology

Vision

To be a premier institute of professional education and research, responsive to the needs of industry and society.

Mission

To achieve academic excellence through innovative teaching- learning practice, by providing conducive research environment, industry-institute interaction and skill development, leading to professionals with ethical values and social responsibilities.

Department of Information Science and Engineering

Vision

To be a centre of excellence in Information Science & Engineering with quality education and research, responsive to the needs of industry and society.

Mission

- To achieve academic excellence through innovative teaching-learning practice.
- To inculcate the spirit of innovation, creativity and research.
- To enhance employability through skill development and industry-institute interaction.
- To develop professionals with ethical values and social responsibilities.

Program Educational Objectives (PEOs)

Graduates will be

PEO1: Competent professionals with knowledge of Information Science & Engineering to pursue variety of careers and higher education.

PEO2: Proficient in designing innovative solutions to real life problems that are technically sound, economically viable and socially acceptable.

PEO3: Capable of working in teams, adapting to new technologies and upgrading skills required to serve the society with ethical values.

FILE STRUCTURES LABORATORY WITH MINI PROJECT
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2017 - 2018)
SEMESTER – VI

Subject Code	18ISL67	IA Marks	40
Number of Lecture Hours/Week	01I + 02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03

CREDITS – 02

Description (If any):

Design, develop, and implement the following programs

Lab Experiments:

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.

Course outcomes: The students should be able to:

- Implement operations related to files
- Apply the concepts of file system to produce the given application.
- Evaluate performance of various file systems on given parameters.

Conduction of Practical Examination:

1. All laboratory experiments from part A are to be included for practical

examination.

2. Mini project has to be evaluated for 30 Marks as per 6(b).
3. Report should be prepared in a standard format prescribed for project work.
4. Students are allowed to pick one experiment from the lot.
5. Strictly follow the instructions as printed on the cover page of answer script.
6. Marks distribution:
   a) Part A: Procedure + Conduction + Viva: **04 + 21 +05 =30 Marks**
   b) Part B: Demonstration + Report + Viva voce = **10+49+11 = 70 Marks**
7. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

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

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”

Pipes:
A pipe is nothing but a temporary storage place where the output of one command is stored and then passed as the input for second command. Pipes are used to run more than two commands (multiple commands) from same command line.

- 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
```

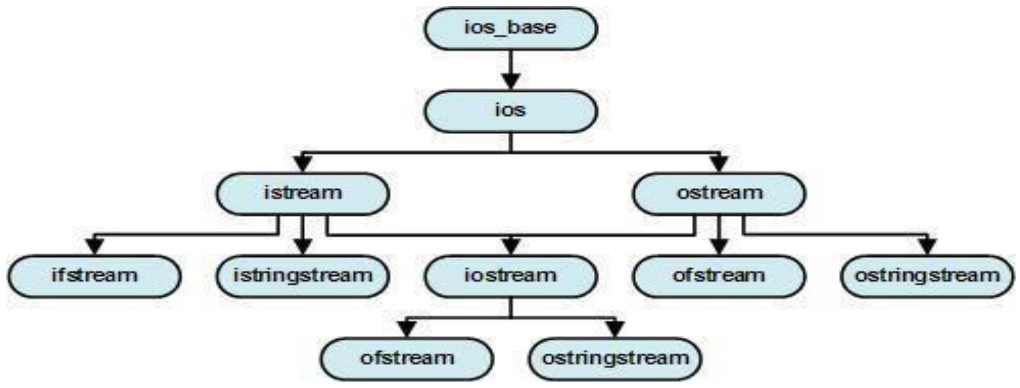
File I/O:
Perform output and input of characters to or from files

Standard 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.



You can first declare an instance of the **stream** class using one of its constructors from the following syntaxes to open a file :

- **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.

Syntaxes :

ofstream obj(const char* FileName, int FileMode); or
ifstream obj(const char* *FileName*, int *FileMode*); or
fstream obj(const char* *FileName*, int *FileMode*);

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::trunk	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

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);

tellg() and tellp()

These two member functions have no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer (in the case of tellg) or the put stream pointer (in the case of tellp).

seekg() and seekp()

These functions allow us to change the position of the get and put stream pointers. Both functions are overloaded with two different prototypes. The first prototype is:

```
seekg ( position );
seekp ( position );
```

Using this prototype the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is the same as the one returned by functions tellg and tellp: the member type pos_type, which is an integer value.

The other prototype for these functions is:

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

Using this prototype, the position of the get or put pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off_type, which is also an integer type. Anddirection is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from, and that can take any of the following values:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position of the stream pointer
ios::end	offset counted from the end of the stream

(In Command prompt, type “gedit Program1.cpp” and press Enter button, to open the editor. Type the following code.)

Program1.cpp :

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

using namespace std;

char * strrev(char *str1)
{
        int i=0;
        int len=strlen(str1);
        char *str2=new char[20];
        while(len>0)
        {
                str2[i]=str1[--len];
                i++;
        }
        str2[i]='\0';
        return str2;
}

main()
{
        int choice,i,num_name;
        fstream fp,fp1;
        char name[20][20],filename[20],str[10],filename1[20];
        while(1)
        {
                cout<<"1 : Standard input to standard output\n";
                cout<<"2 : file to standard output\n";
                cout<<"3 : file to file\n 4: exit\n";
                cout<<"Enter your choice :";
                cin>>choice;
                switch(choice)
                {
```

```
        case 1:cout<<"Enter the number of names to read :\n";
                cin>>num_name;
                for(i=1;i<=num_name;i++)
                        cin>>name[i];
                cout<<"\nReversed names are :\n";
                for(i=1;i<=num_name;i++)
                {
                        cout<<strrev(name[i])<<"\n";;
                }
                break;

      case 2: cout<<"Enter the input file name\n";

                cin>>filename;
                fp.open(filename,ios::in);
                if(fp.fail())
                {
                        cout<<"cannot open the file\n";
                        return 1;
                }
                else
                {
                        cout<<"\nReversed names from the file:\n";
                        while(1)
                        {
                                fp>>str;
                                if(fp.fail())
                                        break;
                                cout<<strrev(str)<<endl;
                        }
                       fp.close();
                }
                break;
        case 3:cout<<"Enter the input file name\n";
                cin>>filename;
                cout<<"Enter the output file name\n";
                cin>>filename1;
                fp.open(filename,ios::in);
                fp1.open(filename1,ios::out);
```

```
                    if(fp.fail()||fp1.fail())
                    {
                        cout<<"Can not open the file\n\n";
                        return 1;
                    }
                    while(1)
                    {
                        fp>>str;
                        if(fp.fail())
                            break;
                        fp1<<strrev(str)<<"\n";

                    }
                    fp.close();
                    fp1.close();
                    break;
                default:exit(0);
            }
        }
}
```

Output :

```
$ g++ Program1.cpp
$ ./a.out
1 : Standard input to standard output
2 : file to standard output
3:file to file
 4: exit
Enter your choice :1
Enter the number of names to read :
2
anil
anusha

Reversed names are :
lina
ahsuna
```

```
1 : Standard input to standard output
2 : file to standard output
3:file to file
 4: exit
Enter your choice :2
Enter the input file name
1.txt

Reversed names from the file:
1|aydiv
9|anahcar
1 : Standard input to standard output
2 : file to standard output
3:file to file
 4: exit
Enter your choice :3
Enter the input file name
student.txt
Enter the output file name
studentr.txt
```

Note :

1) open the input file and output file to check file contents.

```
gedit student.txt
gedit student.txt
```

2) Before running the program, create input file with some name and some names typed one per line as the contents of that file. Create an output file with some name and with nocontents.

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

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

Delimited Variable Length Fields

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

1. The fields within a record are followed by a delimiting byte or series of bytes.
2. Fields within a record can have different sizes.
3. Different records can have different length fields.
4. Programs which access the record must know the delimiter.
5. The delimiter cannot occur within the data.
6. If used with delimited records, the field delimiter must be different from the record delimiter.
7. There is external overhead for field separation equal to the size of the delimiter per field.
8. 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 percular record taken from the file in buffer.

Program2.cpp :

```
#include<iostream>
#include<fstream>
#include<string.h>
#include<stdlib.h>
#define SIZE 50
using namespace std;
fstream file;
class fixedf
{
        struct student
        {
                char usn[11],name[15],sem[6],dept[6];
        };
        public: void pack();
                void unpack();
                void  search();
                void modify();
};

void fixedf::pack()
{
        char b[SIZE+1];
        student s;
        cout<<"\n Enter usn, name, sem, dept:";
        cin>>s.usn>>s.name>>s.sem>>s.dept;
        file.open("student.txt",ios::out|ios::app);
        sprintf(b,"%s|%s|%s|%s|", s.usn,s.name,s.sem, s.dept);
        int len=strlen(b);
        while(len<(SIZE))
        {
                strcat(b,"-");
                len++;
        }
        strcat(b,"$");
        file<<b;
        file.close();
}
```

```
void fixedf::search()
{
        char temp[50],b[SIZE+1],usn[11];
        student s;
        file.open("student.txt",ios::in);
        cout<<"Enter USN to be searched:";
        cin>>usn;
        while(!file.eof())
        {
                file.getline(b,100,'$');
                //sscanf(b,"%s|%s|%s|%s|",s.usn,s.name,s.sem,s.dept);
                sscanf(b,"%[^|]|%[^|]|%[^|]|%[^|]",s.usn,s.name,s.sem,s.dept);
                if(strcmp(s.usn,usn)==0)
                {
                cout<<"\nRecord found\n";
                cout<<s.usn<<""<<s.name<<""<<s.sem<<""<<s.dept<<endl;
                return;
                }
        }
        cout<<"\n Record not found";
        return;
}

void fixedf::modify()
{
        char b[SIZE+1],usn[11];
        student s;
        int n=0,ch;
        file.open("student.txt",ios::in|ios::out);
        cout<<"\n Enter USN to be modified :";
        cin>>usn;
        while(!file.eof())
        {
                file.getline(b,100,'$');
                //sscanf(b,"%s|%s|%s|%s|",s.usn,s.name,s.sem,s.dept);
                sscanf(b,"%[^|]|%[^|]|%[^|]|%[^|]",s.usn,s.name,s.sem,s.dept);
                if(strcmp(s.usn,usn)==0)
                {
                        cout<<"\n KEY found\n";
```

```
                    cout<<"\n 1:Modify name 2: Modify sem 3: Modify Department \n";
                    cout<<"\n Enter your choice:";
                    cin>>ch;
                    switch(ch)
                    {
                            case 1: cout<<"\nEnter name:";cin>>s.name;
                                    break;
                            case 2: cout<<"\nEnter sem:"; cin>>s.sem;
                                    break;
                            case 3: cout<<"\nEnter dept:"; cin>>s.dept;
                                    break;
                            default : break;

                    }
                    sprintf(b, "%s|%s|%s|%s|",s.usn,s.name,s.sem,s.dept);
                    int len=strlen(b);
                    while(len<(SIZE))
                    {
                            strcat(b,"-");
                            len++;

                    }
                    strcat(b,"$");
                    file.seekp((n*(SIZE+1)),ios::beg);
                    file<<b;
                    return;
            }
            n++;
        }

        cout<<"\n Record not found";
        return;
}

void fixedf::unpack()
{
        char b[SIZE+1];
        student s;
        fstream file;
        file.open("student.txt", ios::in);
        while(!file.eof())
        {
                file.getline(b,52,'$');
                if(file.eof())
                        return;
                //sscanf(b,"%s|%s|%s|%s|",s.usn,s.name,s.sem,s.dept);
                sscanf(b,"%[^|]|%[^|]|%[^|]|%[^|]",s.usn,s.name,s.sem,s.dept);
                cout<<s.usn<<""<<s.name<<""<<s.sem<<""<<s.dept<<endl;
        }
}

main()
{
        fixedf f;
        int ch;
        while(1)
        {
                cout<<"\n 1:pack 2:unpack 3:search 4:modify 5:exit\n";
                cout<<"\n Enter your choice :";
                cin>>ch;
                switch(ch)
                {
                       case 1: f.pack();
                               break;
                       case 2: f.unpack();
                               file.close();
                               break;
                        case 3: f.search();
                               file.close();
                               break;
                        case 4: f.modify();
                               file.close();
                               break;
                        default: exit(0);
                }
        }
}
```

Output :

```
$ g++ Program2.cpp
$ ./a.out

 1:pack 2:unpack 3:search 4:modify 5:exit
 Enter your choice :1
 Enterusn, name, sem, dept:44 kiran 5 IS
 1:pack 2:unpack 3:search 4:modify 5:exit

 Enteryourchoice:2
4sn14anupama1cs
1snaryaissit
54raj5cs
43vishu5is
67dhanu6is
80shama2IS
81bhavish4IS
44kiran5IS

Enter your choice :4
 Enter USN to be modified :54
 KEY found
 1:Modify name 2: Modify sem 3: Modify Department
 Enter your choice:1
Enter name:raju
 1:pack 2:unpack 3:search 4:modify 5:exit

 Enter your choice :3
Enter USN to be searched:54
Record found
54raju5cs
```

3. Write a program to read and write and 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.

Program3.cpp :

```
#include<iostream>
#include<string.h>
#include<fstream>
#include<stdio.h>
#include<stdlib.h>
#define RECORDSIZE 50
using namespace std;
fstream file;
class  varlen_student
{
	struct student
	{
		char usn[11],name[15],branch[15],college[15];
	};
	public:
		void pack();
		void unpack();
		void search();
		void modify();
};

void varlen_student::pack()
{
	student s;
	char buff[RECORDSIZE+1];
	cout<<"enter usn,name,branch and college\n";
	cin>>s.usn>>s.name>>s.branch>>s.college;
	sprintf(buff,"%s|%s|%s|%s|$",s.usn,s.name,s.branch,s.college);
	file.open("pg.txt",ios::out|ios::app);
	file<<buff;
	file.close();
}

void varlen_student::unpack()
{
	student s;
	char buff[RECORDSIZE+1];
```

```
        file.open("pg.txt",ios::in);
while(!file.eof())
        {
                file.getline(buff,RECORDSIZE,'$');
                if(file.eof())
                        break;
                sscanf(buff,"%[^|]|%[^|]| %[^|]|%[^|]|$",
s.usn,s.name,s.branch,s.college);
                cout<<"\n----------------
   \nusn\t:"<<s.usn<<"\nname\t:"<<s.name<<"\nbranch\t:"
                   <<s.branch<<"\ncollege:"<<s.college<<endl;
        }
        file.close();
}
void varlen_student::search()
{
        student s;
        int found=0;
        char buff[RECORDSIZE+1];
        char usn[11];
        file.open("pg.txt",ios::in);
        cout<<"\nEnter the usn to be searched:";
        cin>>usn;
        while(!file.eof() && found==0)
        {
                file.getline(buff,RECORDSIZE,'$');
                if(file.eof())
                        break;

        sscanf(buff,"%[^|]|%[^|]|%[^|]|%[^|]|$",s.usn,s.name,s.branch,s.college);
                if(strcmp(s.usn,usn)==0)
                {
                        found=1;
                        cout<<"\nRecord found";
                        cout<<"\n ------------ \nusn\t:"<<s.usn<<"\nname\t:"<<s.name
                        <<"\nbranch\t:"<<s.branch<<"\ncollege:"<<s.college<<endl;
                        break;
                }
        }
        if(found==0)
                cout<<"\nRecord not found\n";
        file.close();
}

void varlen_student::modify()
{
        int n=0,found=0;
        student s;
        char usn[11];
        char buff[RECORDSIZE+1];
        fstream newfile;
        file.open("pg.txt",ios::in);
newfile.open("temp.txt",ios::out|ios::app);
        cout<<"\nEnter the usn to be modified:";
        cin>>usn;
        while(!file.eof())
        {
                file.getline(buff,RECORDSIZE,'$');
                if(file.eof())
                        break;

        sscanf(buff,"%[^|]|%[^|]|%[^|]|%[^|]|$",s.usn,s.name,s.branch,s.college);
                if(strcmp(s.usn,usn)==0)
                {
                        found=1;
                        cout<<"\nRe-Enter student details: ";
                        cout<<"enter usn,name,branch and college\n";
                        cin>>s.usn>>s.name>>s.branch>>s.college;
                }
                sprintf(buff,"%s|%s|%s|%s|$",s.usn,s.name,s.branch,s.college);
                newfile<<buff;
        }
        if(!found)
                cout<<"\n\nRecord to be modified does not exist in file\n";
        file.close();
        newfile.close();
        remove("pg.txt");
        rename("temp.txt","pg.txt");
```

```
}
int main()
{
	int ch;
	varlen_student b;
	for(;;)
	{
		cout<<"\n1.pack\t2.unpack\t3.search\t4.modify\t5.exit\nEnteryour
choice:";
		cin>>ch;
		switch(ch)
		{
			case 1: b.pack();
				break;
			case 2:b.unpack();
				break;
			case3:b.search();
				break;
			case 4: b.modify();
				break;
			default:exit(0);
		}
	}
}
```

Output :

```
$ g++ program3.cpp
$ ./a.out


1.pack2.unpack    3.search     4.modify     5.exit
Enter your choice:1
enter usn,name,branch and college
4sn16is001 anusha IS SIT

1.pack2.unpack    3.search     4.modify     5.exit
Enter your choice:2
----------------
```

```
usn    :4sn16is001
name :anusha
branch :IS
college:SIT
1.pack2.unpack    3.search      4.modify      5.exit
Enter your choice:3
Enter the usn to be searched:22

Record not found

1.pack2.unpack    3.search      4.modify      5.exit
Enter your choice:3
Enter the usn to be searched:4sn16is001

Record found
---------------
usn    :4sn16is001
name :anusha
branch :IS
college:SIT

1.pack2.unpack    3.search      4.modify      5.exit
Enter your choice:4

Enter the usn to be modified:123


Record to be modified does not exist in file

1.pack2.unpack    3.search      4.modify      5.exit
Enter your choice:4

Enter the usn to be modified:4sn16is001

Re-Enter student details: enter usn,name,branch and college
4sn16is001 Advi CSSIT

1.pack 2.unpack    3.search      4.modify      5.exit
Enter your choice:2

---------------
usn    :4sn16is001
name :Advi
```

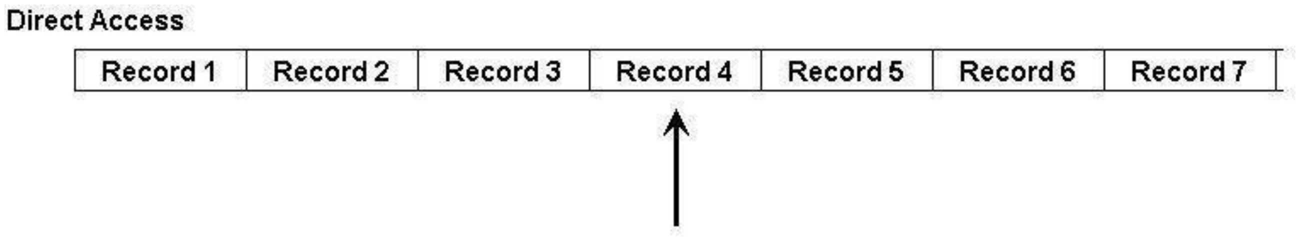
```
branch         :CS
college:SIT

1.pack2.unpack    3.search      4.modify      5.exit
Enter your choice:5
```

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 lenth record



Program4.cpp:

```
#include<iostream>
#include<stdio.h>
#include<fstream>
#include<stdlib.h>
#include<string.h>
using namespace std;
class Student
{
        int rrn[10];
        char usn[30];
        char name[30];
        char branch[30];
        char college[30];
        char buffer[100];
        int count;
        public:
        void input();
        void output();
        void searchrrn();
        void creatrrn();
        void pack();
        void unpack();
        void Write();
};

void Student:: input()
{
        cout<<"Enter Usn"<<endl;
        cin>>usn;
        cout<<"Enter Name"<<endl;
        cin>>name;
        cout<<"Enter Branch"<<endl;
        cin>>branch;
        cout<<"Enter College"<<endl;
        cin>>college;
}
```

```
void Student:: output()
{
	cout<<"Usn :";
	puts(usn);
	cout<<"Name :";
	puts(name);
	cout<<"Branch :";
	puts(branch);
	cout<<"College :";
	puts(college);
}

void Student::pack()
{
	strcpy(buffer,usn); strcat(buffer,"|");
	strcat(buffer,name); strcat(buffer,"|");
	strcat(buffer,branch); strcat(buffer,"|");
	strcat(buffer,college); strcat(buffer,"|");
	strcat(buffer,"#");
}
void Student::unpack()
{
	char *ptr = buffer;
	while(*ptr!='#')
	{
		if(*ptr == '|')
		*ptr ='\0';
		ptr++;
	}
	ptr = buffer;
	strcpy(usn,ptr);
	ptr=ptr+strlen(ptr)+1;
	strcpy(name,ptr);
	ptr=ptr+strlen(ptr)+1;
	strcpy(branch,ptr);
	ptr = ptr+strlen(ptr)+1;
	strcpy(college,ptr);

}
```

```
void Student:: Write()
{
        fstream os("rrn.txt",ios::out|ios::app);
        //fstream os;
        //os.open("rrn.txt",ios::out|ios::app);

        os.write(buffer,strlen(buffer));
        os<<endl;
        os.close();
}

void Student::creatrrn()
{
        fstream fs;
        int pos;
        count=-1;
        fs.open("rrn.txt",ios::in);
        while(fs)
        {
                pos=fs.tellg();
                fs.getline(buffer,'#');
                if(fs.eof())
                        break;
                rrn[++count]=pos;
        }
        fs.close();
}

void Student::searchrrn()
{
        int pos=-1;
        int key;
        cout<<"\n ENTER THERRN:";
        cin>>key;
        if(key>count)
                cout<<"\n RECORD IS NOT FOUND\n";
        else
        {
                fstream is("rrn.txt",ios::in);
                pos=rrn[key];
                is.seekp(pos,ios::beg);
                is.getline(buffer,'#');
                unpack();
                output();
                is.close();
        }
}


main()
{
        int choice = 1;
        Student ob;
        while(choice < 3)
        {
                cout<<"1> Insert A Record "<<endl;
                cout<<"2> Search For A Record "<<endl;
                //cout<<"3> Unpack record"<<endl;
                cout<<"3> Exit"<<endl;
                cin>> choice;
                switch(choice)
                {
                        case 1: ob.input();
                                ob.pack();
                                ob.Write();
                                break;

                        case 2: ob.creatrrn();
                                ob.searchrrn();
                                break;
                        //case 3:ob.unpack();
                        case 3:exit(0);
                }
        }
}
```

Output :

```
$ g++ Program4.cpp
$ ./a.out

1> Insert A Record
2> Search For A Record
3> Exit
1
Enter Usn
1
Enter Name
ashwa
Enter Branch
cs
Enter College
sit

1> Insert A Record
2> Search For A Record
3> Exit
1
Enter Usn
2
Enter Name
veena
Enter Branch
cs
Enter College
sit
1> Insert A Record
2> Search For A Record
3> Exit
2

 ENTER THE RRN:0
Usn :1
Name :ashwa
Branch :cs
College :sit
```

```
1> Insert A Record
2> Search For A Record
3> Exit
2

 ENTER THE RRN:1
Usn :2
Name :veena
Branch :cs
College:sit

1> Insert A Record
2> Search For A Record
3> Exit
2
 ENTER THE RRN:3

 RECORD IS NOT FOUND
```

5. **Write a program to implement simple index on primary key for a file of student 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

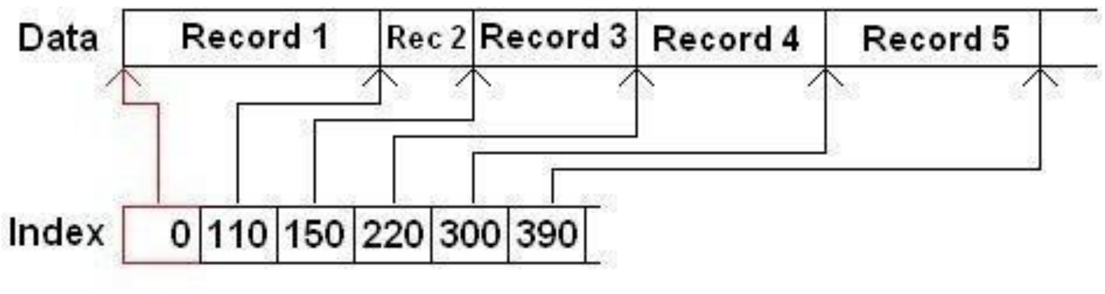
The part of an index which contains information to locate records.

- 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.



Program5.cpp :

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<fstream>
using namespace std;
fstream data,indx;
char pri[125][15];
int ind[125],count=0;
int flag_updt=0;
void sort();
class Index_student
{
        struct student
        {
                char usn[11];
                char name[20];
                char sem[2];
                char branch[5];
                char college[10];
        };
        public:
                void Insert();
                void Delete();
                void Search();
                void LoadIndex();
                void WriteIndex();
};

void Index_student::Insert()
{
                char buf[100];
                int pos;
                student s;
                data.open("std.txt",ios::out|ios::app);
                data.seekg(0,ios::end);
                pos=data.tellg();
                cout<<" ----------------------- \n";
                cout<<"ENTER RECORD DETAILS\n";
                cout<<" -------------------------- \n";
                cout<<"USN : ";

                cin>>s.usn;
                for(int i=0;i<count;i++)
                {
                        if(strcmp(pri[i],s.usn) == 0)
                        {
                                cout<<endl;
                                cout<<"DUPLICATE RECORD !!!";
                                data.close();
                                return;
                        }
                }
                cout<<"Name    :";
                cin>>s.name;
                cout<<"Sem    : ";
                cin>>s.sem;
                cout<<"Branch :";
                cin>>s.branch;
                cout<<"College:";
                cin>>s.college;
                sprintf(buf,"|%s|%s|%s|%s|%s#",s.usn,s.name,s.sem,s.branch,s.college);


                data<<buf;
                strcpy(pri[count],s.usn);
                ind[count]=pos;
                count++;
                flag_updt=1;
                data.close();
}

void Index_student::Delete()
{
        char reg[20],buf[100],usn[15];
        int pos;
        int flag=0;
        cout<<"ENTER USN:";
        cin>>reg;
        data.open("std.txt",ios::in|ios::out);
        for(int i=0;i<count;i++)
                if(strcmp(pri[i],reg) == 0)
                {
                        flag=1;
                        data.seekg(ind[i],ios::beg);
                        data.getline(buf,100,'#');
                        buf[0]='*';
                        pri[i][0]='*';
                        data.seekg(ind[i],ios::beg);

                        data<<buf;
                        cout<<"RECORD DELETED\n\n";
                        flag_updt=1;
                        break;
                }
        if(flag==0)
        {

                cout<<"RECORD NOT FOUND!!!\n\n";
        }
        data.close();
}

void Index_student::Search()
{
        int flag=0;
        char buf[100];
        char usn[20];
        student s;
        cout<<"ENTER USN: ";
        cin>>usn;
        data.open("std.txt",ios::in|ios::out);
        for(int i=0;i<count;i++)
        {
                if(strcmp(pri[i],usn) == 0)
                {
                        flag=1;
                        data.seekg(ind[i]);
                        data.getline(buf,100,'#');

        sscanf(buf,"|%[^|]|%[^|]|%[^|]|%[^|]|%[^|]#",s.usn,s.name,s.sem,s.branch,s.c
ollege);
                        cout<<" ---------------------- ";
                        cout<<endl<<"RECORD DETAILS";
                        cout<<endl<<" ----------------------";
                        cout<<endl<<"USN        : "<<s.usn;
                        cout<<endl<<"NAME       : "<<s.name;
                        cout<<endl<<"SEMESTER   : "<<s.sem;
                        cout<<endl<<"BRANCH     : "<<s.branch;
                        cout<<endl<<"COLLEGE    : "<<s.college;
                        cout<<"\n";
                }
        }
        if(flag==0)
                cout<<"RECORD DOES NOT EXISTS\n\n";
        data.close();
}

void Index_student::LoadIndex()
{
        char buf[100],temp[100];
        indx.open("indx.txt",ios::in);
        while(indx)
        {
                indx.getline(buf,100,'#');
                if(indx.eof())
                        break;
                sscanf(buf,"|%[^|]|%[^|]",pri[count],temp);
                ind[count]=atoi(temp);
                count++;
        }
        indx.close();
}

void Index_student::WriteIndex()
```

```
{
        char buf[100];
        indx.open("indx.txt",ios::out);
        sort();
        for(int i=0;i<count;i++)
        {
                sprintf(buf,"%d",ind[i]);
                indx<<"|"<<pri[i]<<"|"<<buf<<"|"<<"#";
        }
        indx.close();
}
 void sort()
 {
 char temp[20];
 int tempid, i,j;
 for(i=0;i<count;i++)
 {
         for(j=i+1;j<count;j++)
         {
                 if(strcmp(pri[i],pri[j])>0)
                 {
                         strcpy(temp,pri[i]);
                         strcpy(pri[i],pri[j]);
                         strcpy(pri[j],temp);
                         tempid=ind[j];
                         ind[j]=ind[i];
                         ind[i]=tempid;
                 }
         }
      }
 }

main()
{       Index_student I;
        I.LoadIndex();
        int ch;
        while(1)
        {
                cout<<"1->INSERT  RECORD\n";
```

```
                    cout<<"2->DELETE RECORD\n";
                    cout<<"3->SEARCH RECORD\n";
                    cout<<"4->QUIT\n";
                    cout<<"ENTER YOUR CHOICE:";
                    cin>>ch;
                    switch(ch)
                    {
                              case 1:I.Insert();      I.WriteIndex();
                                          break;
                               case 2:I.Delete();      I.WriteIndex();
                                        break;
                              case 3:I.Search();
                                        break;

                              case 4: exit(0);
                    }
          }
}
```

Output :

```
$ g++ program5.cpp
$ ./a.out

1->INSERT  RECORD
2->DELETE  RECORD
3->SEARCH RECORD
4->QUIT
ENTER YOUR CHOICE:1
----------------------------
ENTER RECORD DETAILS
---------------------------
USN : 10
Name   :uma
Sem     1
Branch : cs
College : sit
1->INSERT RECORD
2->DELETE RECORD
3->SEARCH RECORD
4->QUIT
ENTER YOUR CHOICE:1
----------------------------
ENTER RECORD DETAILS
---------------------------
USN : 9
Name   :usha
Sem     1
Branch : is
College : sit
1->INSERT RECORD
2->DELETE RECORD
3->SEARCH RECORD
4->QUIT
ENTERYOUR CHOICE:3
ENTER USN:2
RECORD DOES NOTEXISTS

1->INSERT RECORD
```

```
2->DELETE RECORD
3->SEARCH RECORD
4->QUIT
ENTERYOUR CHOICE:3
ENTER USN:9
--------------------------
RECORD DETAILS
-------------------------
USN       : 9
NAME      :usha
SEMESTER    : 1
BRANCH    : is
COLLEGE   : sit
1->INSERT RECORD
2->DELETE RECORD
3->SEARCH RECORD
4->QUIT
ENTERYOUR CHOICE:2
ENTER USN:10
RECORD DELETED

1->INSERT RECORD
2->DELETE RECORD
3->SEARCH RECORD
4->QUIT
ENTERYOUR CHOICE:3
ENTER USN:10
RECORD DOES NOTEXISTS
```

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

Index: a table containing key and its address
Key: an entry in index table which uniquely identifies the record.
Secondary key: an entry in the index table (not unique, repetition of the key may occur) which identifies the record.

- Operations performed on index file
- Create the original empty index and data files
- Load the index file into memory before using it
- Rewrite the index file after using it
- Add data records to the data file
- Delete records from the data file
- Update the index file to reflect the changes in the data file

Program6.cpp :

```
#include<iostream>
#include<fstream>
#include<string>
#include<sstream>
#include<stdlib.h>
using namespace std;
class student
{
 public:
        string usn;
        string name;
        string branch;
        string sem;
        string buffer;
      string Name_list[100];
      int Address_list[100];
      int count;
        student(){ count=-1;}
        void read_data();
        void pack();
        void write_to_file();
        void disp();
      void remove(string);
      void delete_from_file(int);
      void search(string);
      int   search_index(string);
      void  read_from_file(int);
      void sort_index();
};
void student::read_data()
{
      cout<<"usn:";
      cin>>usn;
      cout<<"name:";
      cin>>name;
      cout<<"branch:";
        cin>>branch;
      cout<<"semester:";
```

```
  cin>>sem;
}
void student::pack()
{
        buffer.erase();
        buffer+=usn+"|"+name+"|"+branch+"|"+sem+"$"+"\n";
}


void student::write_to_file()
{
        int pos;
        fstream file;
        file.open("6a.txt",ios::out|ios::app);
        pos=file.tellp();
        file<<buffer;
        file.close();
        Name_list[++count]=name;
        Address_list[count]=pos;
        sort_index();
}
void student::disp()
{
        int i;
        cout << endl << "INDEX FILE " << endl;
        for(i=0;i<=count;i++)
        cout<<endl<<Name_list[i]<<" "<<Address_list[i];
        cout<<"\n";
        system("cat 6a.txt");
}

void student::sort_index()
{
        int i,j,temp_Address;
        string temp_Name;
        for(int i=0;i<=count;i++)
        {
                for(int j=i+1;j<=count;j++)
                {
                        if(Name_list[i]>Name_list[j])
                        {
                        temp_Name=Name_list[i];
                        Name_list[i]=Name_list[j];
                        Name_list[j]=temp_Name;

                                temp_Address=Address_list[i];
                        Address_list[i]=Address_list[j];
                        Address_list[j]=temp_Address;
                        }
                }
        }
}

int student::search_index(string key)
{
        int low=0,high=count,mid=0,flag=0,pos;
        while(low<=high)
        {
                mid=(low+high)/2;

                if(Name_list[mid]==key){flag=1;break;}
                if(Name_list[mid]>key)high=mid-1;
                if(Name_list[mid]<key)low=mid+1;

        }
        if(flag)
                return mid;
        else
                return -1 ;
}

void student::search(string key)
{
        int pos=0,t;
        string buffer;
        buffer.erase();
        pos=search_index(key);
        if(pos==-1)
        cout << endl << "record not found" << endl;
        else if(pos>=0)
        {
                read_from_file(pos);
                t=pos;
                while(Name_list[++t]==key && t<=count) read_from_file(t);
                t=pos;
                while(Name_list[--t]==key && t>=0) read_from_file(t);

        }
}

void student::read_from_file(int pos)
{
        int address,i;
        fstream file;
        file.open("6a.txt",ios::in|ios::app);
        address=Address_list[pos];
        file.seekp(address,ios::beg);
        buffer.erase();
        getline(file,buffer);
        cout<<"\nFound the record: "<<buffer;
        file.close();
}
void student::remove(string key)
{
        int pos=0,t,choice;
        string buffer;
        buffer.erase();
        pos=search_index(key);
        if(pos==-1)
                cout<<endl<<"not possible to remove";
        else if(pos>=0)
        {
                read_from_file(pos);
                cout<<"\nDelete?(1/0):";
                cin>>choice;
                if(choice)delete_from_file(pos);
                t=pos;
                while(Name_list[++t]==key)
```

```
		{
			read_from_file(t);
			cout<<"\nDelete?";
			cin>>choice;
			if(choice)delete_from_file(t);
		}
		t=pos;

		while(Name_list[--t]==key )
		{
		read_from_file(t);
		cout<<"\nDelete?";
		cin>>choice;
		if(choice)
			delete_from_file(t);
		}

	}

}
void student::delete_from_file(int pos)
{
	int i,address;
	fstream file;
	file.open("6a.txt");
	address=Address_list[pos];
	file.seekp(address,ios::beg);
	file.put('*');
	cout<<"\nRecord Deleted: ";
	for(i=pos;i<count;i++)
	{
	Name_list[i]=Name_list[i+1];
		Address_list[i]=Address_list[i+1];
	}
	count--;
}

int main()
{
	int choice,count,i;
```

```
        string key;
        student s1;
        while(1)
        {
                cout<<"\nMain Menu\n ------ \n1.Add\n2.Search\n3.Delete\n4.Exit\n--\n";
                cout<<"Enter the choice:";
                cin>>choice;
                switch(choice)
                {
                        case 1: cout<<"\nhow many records to insert\n";
                        cin>>count;
                        for(i=0;i<count;i++)
                        {
                                cout<<"data\n";
                                s1.read_data();
                                s1.pack();
                                s1.write_to_file();
                        }
                        break;

                        case 2: system("clear");
                                s1.disp();
                                cout<<"\nEnterthe name\n";
                                cin>>key;
                                s1.search(key);
                                break;
                        case 3:         cout<<"\n\nEnter thename\n";
                                cin>>key;
                                s1.remove(key);
                                break;
                        case 4:         return 0;
                        default:cout<<"\n\nWrong choice\n";
                                break;

                }
        }
}
```

Output :

```
$ g++ program6.cpp
$ ./a.out


Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter the choice:1

how many records to insert
2
data
usn:4sn16is002
name:vishu
branch:is
semester:6
data
usn:4sn16is004
name:uma
branch:is
semester:6

Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter the choice:2

INDEX FILE
uma 23
vishu 0
```

```
4sn16is002|vishu|is|6$
4sn16is004|uma|is|6$

Enterthe name
souju

record not found

Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter the choice:2

INDEX FILE
uma 23
vishu 0
4sn16is002|vishu|is|6$
4sn16is004|uma|is|6$

Enterthe name
vishu

Found the record: 4sn16is002|vishu|is|6$
Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter thechoice:3
Enter the name
vinu

not possible to remove
```

```
Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter thechoice:3
Enter the name
vishu

Found the record: 4sn16is002|vishu|is|6$
Delete?(1/0):1
Record Deleted:
Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter thechoice:2
INDEX FILE
uma 23
*sn16is002|vishu|is|6$
4sn16is004|uma|is|6$

Enter the name
vishu
record not found

Main Menu
--------
1.Add
2.Search
3.Delete
4.Exit
---------
Enter the choice:4
```

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.

Cosequential 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 Cosequential 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.

Cosequential 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 this 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.)

Program7.cpp

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
class coseq
{
 private:
      string list1[100],list2[100];
      int count1,count2;
 public:
     void load_list();
      void sort_list();
      voidmatch();
};

void coseq::load_list()
{
           fstreamfile;
           string name;
           count1=-1;count2=-1;
           file.open("name1.txt");
           while(!file.eof())
           {
              name.erase();
              getline(file,name);
              list1[++count1]=name;
          }
          file.close();
          file.open("name2.txt");
          while(!file.eof())
          {
                   name.erase();
                   getline(file,name);
                   list2[++count2]=name;
          }
          file.close();
}
void coseq::sort_list()
```

```
{
        int i,j;
        string temp;
        for(i=0;i<count1;i++)
        {
                for(j=i+1;j<count1;j++)
                {
                        if(list1[i]>list1[j])
                        {

                                temp=list1[i];
                                list1[i]=list1[j];
                                list1[j]=temp;

                        }
                }
        }
         for(i=0;i<count2;i++)
 {
                for(j=i+1;j<count2;j++)
                {
                        if(list2[i]>list2[j])
                        {
                                temp=list2[i];
                                list2[i]=list2[j];
                                list2[j]=temp;
                        }
                }
        }
}
void coseq::match()
{
         int i=0,j=0,flag=0;
         while(i<count1 && j<count2)
        {
                if(list1[i]==list2[j])
                {
                        cout<<list1[i]<<"\n";
                         i++;
```

```
                                j++;
                                flag=1;
                      }
                      if(list1[i]<list2[j])
                     i++;
                      if(list1[i]>list2[j])
                      j++;
       }
       if(flag==0) cout<<"no matchfound\n";
}

int main()
{
coseqc1;
           c1.load_list();
           c1.sort_list();
           c1.match();
           return 0;
}
```

/*Create Text File*/

```
$gedit name1.txt

Gousekhan
Hithashree
Divya

$ gedit name2.txt

Jayashree
Aishwarysr
Divya
```

Output:

```
$ g++ program7.cpp
$ ./a.out
Divya
$ ./a.out
no match found
```

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.

Program8.cpp :

```
#include<iostream>
#include<stdio.h>
#include<fstream>
#include<stdlib.h>
#include<string.h>
using namespace std;
class Merge
{
        char list[9][30][20];
        char outputlist[270][20];
        int k,count;
        int size[9];
        int index[9];
        public:
        void sort(char l[30][20],int s);
        void merge();
        void read(char temp[30]);
        void load(char filename[20]);
        char* minValue();
        void display();
};
void Merge::read(char temp[30])
{
        int n;
        char name[20];
        fstream fs;
        fs.open(temp,ios::out|ios::app);
        cout<<"Enter hoW many name for list:";
        cin>>n;
        for(int i=0;i<n;i++)
        {
                fflush(stdin);
                cin>>name;
                fs<<name<<endl;
        }
        fs.close();
}
```

```
void Merge :: load(char filename[20])
{
        k = 8;
        char temp[30],buf[10];
        fstream file;
        memset(size,0,sizeof(size));
        memset(index,0,sizeof(index));
        for(int i=1;i<=k;i++)
        {
                strcpy(temp,filename);
                sprintf(buf,"%d",i);
                strcat(temp,buf);
                strcat(temp,".txt");
                read(temp);
                file.open(temp,ios::in);
                while(!file.eof())
                {
                        file.getline(temp,'\n');
                        if(file.eof())
                        break;
                        strcpy(list[i][size[i]],temp);
                        size[i]++;
                }
                file.close();
                sort(list[i],size[i]);
        }
}
void Merge :: merge()
{
        count = 0;
        char *value = minValue();
        while(value != NULL)
        {
                for(int i = 1; i<= k; i++)
                {
                        if(index[i]>= size[i])
                                continue;
                        if(strcmp(value,list[i][index[i]]) == 0)
                                index[i]++;
                }
                strcpy(outputlist[count],value);
                count++;
                value = minValue();
        }
}
char* Merge :: minValue()
{
        int t = 1;
        char *value = NULL;
        while(index[t] >= size[t] && t <= k)
                t++;
        if( t <= k)
        {
                value=list[t][index[t]];
                for(int i=t+1;i<=k;i++)
                        if(strcmp(value,list[i][index[i]]) > 0 && index[i]<size[i])
                                value = list[i][index[i]];
        }
return value;
}


void Merge :: sort(char l[30][20],int s)
{
        char temp[20];
        for(int i= 0;i< s;i++)
        {
                for(int j=i+1; j<s;j++)
                {
                        if(strcmp(l[i],l[j]) > 0)
                        {
                                strcpy(temp,l[i]);
                                strcpy(l[i],l[j]);
                                strcpy(l[j],temp);
                        }
                }
        }
}
```

```
void Merge ::display()
{
        for(int i= 0; i < count; i++)
                cout<<outputlist[i]<<" ";
}

 main()
{
        Merge m;
        char filename[]="list";
//clrscr();
        m.load(filename);
        m.merge();
        cout<<"\nNames in sorted order is:";
        m.display();
//getch();
}
```

Output :

```
 $ g++ Program8.cpp
 $ ./a.out

 Enter how many name for list: 3
neema
navya
kavya
Enter how many name for list: 2
naada
gaana
Enter how many name for list: 2
vinaya
neema
Enter how many name for list: 2
vinaya
gaana
Enter how many name for list: 2
nithya
```

```
kavya
Enter how many name for list: 2
jaya
rekha
Enter how many name for list: 2
sushma
souju
Enter how many name for list: 2
kavya
sapna

Names in sorted order is : gaana jaya kavya naada navya neema nithya rekha sapna
souju sushma vinaya
```

VIVA QUESTIONS

1. What is File Structure?
2. What is a File?
3. What is a field?
4. What is a Record?
5. What is fixed length record?
6. What is RRN?
7. What is Variable length record?
8. What are the different modes of opening a file?
9. What is ifstream()?
10. What is ofstream()?
11. What is the difference between read() and getline()?
12. How to close a file? What happens if a file is not closed?
13. What is Hashing? What is its use?
14. Explain any one collision resolution technique.
15. What is Btree? What is B+tree?
16. Differentiate between Logical and Physical file
17. What is the use of seekg() and seekp()?
18. Explain the different way of write data to a file.
19. What is meant by Indexing?
20. What is multilevel indexing?
21. What is File descriptor?
22. What is Fragmentation? What is internal fragmentation?

23. What is a delimeter?
24. Define direct access.
25. Define sequential access.
26. What is the need of packing the record before writing into the file?
27. Explain ios::trunk and ios::nocreate
28. What is the use of End-of-file (EOF)?
29. What are stdin, stdout and stderr?
30. What is data compression?
31. What are the properties of B tree?
32. How do we delete fixed length records?
33. How can we reclaim the deleted space dynamically?
34. What are the advantages and disadvantages of indexes that are too large to hold in memory?
35. What is an AVL tree?
36. 39. H M L B Q S T N A Z P E G C V J K D I U Show B tree creation, insertion, splitting, deletion, merging and redistribution.
37. What is sprintf() ? Explain its parameters.
38. What is the use of tellg() and tellp()?

Srinivas Institute of Technology

Department of Information Science and Engineering

Program Outcomes (POs)

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 analyse 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 for complex problems.
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 lifelong learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1: Programming and software development skills: Ability to employ modern computer languages, computing environments and standard practices for analysing, designing and developing optimal solutions to deliver quality software products.

PSO2: Domain specific skills: Ability to apply techniques to develop computer based solutions in various domains like Artificial Intelligence, Machine Learning, Network Engineering, Image Processing, Web Technologies and Data Sciences.