# PESIT- SOUTH CAMPUS

**1Km before Electronic City, Hosur Road, Bangalore-560100.**

**DEPARTMENT OF INFORMATION SCIENCE& ENGINEERING**

**VI SEMESTER**

**LAB MANUAL**

**SUBJECT: File StructuresLab with Mini Project**

**SUBJECT CODE: 15ISL68**

**FACULTY : Mrs. Kakoli Bora**

**SESSION: Jan 2018 – May 2018**

# FS LAB MANUAL

**PART - A**
**Design, develop, and implement the following programs**

1. Write a C++ program to read series of names, one per line, fromstandard input and write these names spelled in reverse order to thestandard output using I/O redirection and pipes. Repeat theexercise using an input file specified by the user instead of thestandard input and using an output file specified by the userinstead of the standard output.
2. Write a C++ program to read and write student objects with fixedlengthrecords and the fields delimited by "|". Implement pack ( ),unpack ( ), modify ( ) and search ( ) methods.
3. Write a C++ program to read and write student objects withVariable - Length records using any suitable record structure.Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.
4. Write a C++ 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 C++ program to implement simple index on primary keyfor a file of student objects. Implement add ( ), search ( ), delete ( )using the index.
6. Write a C++ program to implement index on secondary key, thename, for a file of student objects. Implement add ( ), search ( ),delete ( ) using the secondary index.
7. Write a C++ program to read two lists of names and then matchthe names in the two lists using Co-sequential Match based on asingle loop. Output the names common to both the lists.
8. Write a C++ program to read k Lists of names and merge themusing 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.**

/***************************************************************
********
**Problem Statement –**
1. Write a C++ 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.
***************************************************************
*******/

**AIM:** The aim of this program is to read some names from standard input device like keyboard and reverse the names and display them on standard output device like monitor. Repeat the same by reading the names from a file, reverse them and write the reversed names onto an new file.

**Concepts-**

Keywords: I/O redirection, Pipes

**I/O redirection:** There are always three default *files* open, `stdin` (the keyboard), `stdout` (the screen), and `stderr` (error messages output to the screen). These, and any other open files, can be redirected. Redirection simply means capturing output from a file, command, program, script, or even code block within a script and sending it as input to another file, command, program or script.

**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.

**Algorithms-**

1. Enter your choice" 1. From standard I/O 2. From Files"
2. **Case 1:**[using standard I/O]
   Enter the number of names to be read, Read N
   2a. Read one name at a time
   2b. Apply reverse(name) function on each name and display the reversed name on standard O/P and redirect to a file.
   Repeat 2a and 2b for N times
3. **Case 2:** [Using Files]
   `Open the file for reading`
   cout<<"Enter the number of names: ";
   cin>>n;

```
cout<<"The names are:\n";
for(int i=0;i<n;i++)
  {
     Read names from an input file
Reverse the name by calling reverse(name) and write to an output file using
redirection.
  }
Close the file.

[Function reverse( )]
Void reverse(chara[])
{
  int len=strlen(a);
  for(int i=0,j=len-1;i<len/2;++i,--j)
     swap(a[i],a[j]);
}

[Function swap( )]
swap(char &a, char &b)
{
        char temp=a;
        a=b;
        b=temp;
}
```

/******************************************************************************

**Problem Statement –**

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.

*******************************************************************************/

**Aim:** The aim of this problem is to create a text file to store student information using fixed length record where fields of the record is delimited by '|' and to implement the functions search( ) and modify( ) on these records.

**Concepts:**

**Keywords:** Fixed length record, field, delimiter

**Fixed length records:** A data file that contains data records of equal and constant length. All records in the file are the same size. Each field will have a predetermined and unchanging size, set when the record layout is designed, and the sum of the field sizes will add up to the record size. If the data stored in a given field contains fewer characters than the defined size, the rest of the field will be filled with spaces, or some other special characters.

**Fields and delimiter:**Fieldsare the building blocks of records. These are the sections of a record where information is stored.A **delimiter** is a sequence of one or more characters used to specify the boundary between, independent regions in plain text or other data streams(i.e. between two fields of a record or between two records of a file).

```
class student
{
  char name[25],usn[11],age[3],buf[LEN];
  void pack();
  void unpack();
public:
  void add();
  void display();
  void search();
  void modify();
};
```

Function pack( ) packs the fields data(eg. usn, name and age) of a record along with a delimiter '|' between the fields into a buffer. Since it's a fixed length record, the remaining spaces are packed with special character #.

Example: After pack,123|AmesJohn|20#########

```
void student::pack()
{
  strcpy(buf,usn);
  strcat(buf,"|");
  strcat(buf,name);
  strcat(buf,"|");
  strcat(buf,age);
  for(int i=strlen(buf); i<LEN; i++) // LEN is the length of the record fixed
     strcat(buf,"#");
}
```

Function unpack ( ) remove the delimiters between the data of a record  and copy the fields (data) into memory.

```
void student::unpack()
{
  strcpy(usn,strtok(buf,"|"));
  strcpy(name,strtok(NULL,"|"));
  strcpy(age,strtok(NULL,"#"));
}
```

Example: After unpack,usn = 123, name = Ames John, age = 20

**Function add( ) :**This function adds one record to the student file

```
void student::add()
{
  cout<<"Enter USN: ";
  cin>>usn;
  cout<<"Enter name: ";
  cin>>name;
  cout<<"Enter age: ";
  cin>>age;
  pack();
  ofstream fout;
  fout.open("student",ios::app);
  fout<<buf<<endl;
  fout.close();
  cout<<"Student record added\n";
}
```

**Function display( ):** display all the records of the student file

```
void student::display()
{
  ifstream fin;
  fin.open("student");
  while(!fin.eof())
  {
     fin>>buf;
     if(fin.fail())
        break;
     unpack();
     cout<<"\nUSN: "<<usn<<"\nName: "<<name<<"\nAge: "<<age<<endl;
  }
  fin.close();
}
```

**Function search( ):** This function searches for a record in the file based on a given usn using sequential search.

```
        void student::search()
        {
          ifstream fin;
          fin.open("student");
```

```
    char key[4];
    cout<<"Enter the key: ";
    cin>>key;
    while(!fin.eof())
    {
       fin>>buf;
       if(fin.fail())
          break;
       unpack();
       if(!strcmp(usn,key))
       {
          cout<<"\nStudent Found!"<<"\nUSN: "<<usn<<"\nName: "<<name<<"\nAge:
"<<age<<endl;
          fin.close();
          return;
       }
    }
    cout<<"\nStudent not found\n";
    fin.close();
}
```

**Function modify( ):** this function searches a record in the file based on the given usn, if found then modify/change the data of that student otherwise display a message.

```
void student::modify()
{
  fstream f;
  f.open("student",ios::in|ios::out);
  char key[4];
  cout<<"Enter the key: ";
  cin>>key;
  while(!f.eof())
  {
     f>>buf;
     if(f.fail())
        break;
     unpack();
     if(!strcmp(usn,key))
     {
        cout<<"\nStudent found\n"<<"Enter new name: ";
        cin>>name;
        cout<<"Enter new age: ";
        cin>>age;
```

```
          pack();
          f.seekp((int)f.tellg()-LEN,ios::beg);
          f<<buf;
          f.close();
          return;
        }
    }
    cout<<"\nStudent not found\n";
    f.close();
}
```

**Algorithm:**

1. Create a menu for selection of users choice

```
student s;  //object s
do {
cout<<"\n1. Add\n2. Display\n3. Search\n4. Modify\n5.
    Exit\n\nEnter choice: ";
    cin>>ch;
    switch(ch)
    {
    case 1:
       s.add();
       break;
    case 2:
       s.display();
       break;
    case 3:
       s.search();
       break;
    case 4:
       s.modify();
       break;
    case 5:
       break;
    default:
       cout<<"Wrong choice!";
    }
}
while(ch!=5);
```

/***************************************************************************
*******

**Output –**

```
1. Add
2. Display
3. Search
4. Modify
5. Exit

Enter choice: 1
Enter USN: 42
Enter name: Abcdef
Enter age: 21
Student record added

1. Add
2. Display
3. Search
4. Modify
5. Exit

Enter choice: 2

42|Abcdef|21#################
50|abcde|20##################

1. Add
2. Display
3. Search
4. Modify
5. Exit

Enter choice: 4
Enter the key: 42

Student found
Enter new name: Kopparam
Enter new age: 20
```

*************************************************************************/
/********************************************************************************

**Problem Statement –**

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

********************************************************************************/

**Aim:** The aim of this problem is to create a text file to store student information using variable length record where fields of the record is delimited by ‘|’ and to implement the functions search( ) and modify( ) on these records.

**Concepts:**

**Keywords:** Variable length record, field, delimiter

**Variable length records:** A data file that contains data records of variable length. All records in the file are of different size. Each field will have a predetermined size, set when the record layout is designed, and the sum of the field sizes will add up to the record size. Each record will have different size depending on the number of characters entered.

```
classstudent
{
  char name[25],usn[11],age[3],buf[50];
  void pack();
  void unpack();
public:
  void add();
  void display();
  void search();
  void modify();
};
```

Function pack( ) packs the fields data (eg. usn, name and age) of a record along with a delimiter ‘|’ between the fields into a buffer.

Example: **After pack**123|AmesJohn|20#125|Bethoven|21#
Where # is the separator between two record
OR, 123|AmesJohn|20
125|Bethoven|21

```
void student::pack()
{
  strcpy(buf,usn);
  strcat(buf,"|");
  strcat(buf,name);
  strcat(buf,"|");
  strcat(buf,age);
}
```

Function unpack ( ) remove the delimiters between the data of a record and copy the fields (data) into memory.

```
void student::unpack()
{
  strcpy(usn,strtok(buf,"|"));
  strcpy(name,strtok(NULL,"|"));
  strcpy(age,strtok(NULL,"\n")); //OR part of the above example in pack( )
}
```

Example: After unpack  usn = 123, name = Ames John, age = 20

**Function add( ) :** This function adds one record to the student file

1. Read the data (i.e. usn, name, age) of a student
2. Pack the data using pack( ) function
3. Open the file in append mode and write the packed data into it.
4. Close the file.

**Function display( ):** display all the records of the student file

```
1. Open the file in read mode
2. Repeat the following until reached end of the file
   2a. read one record
   2b. unpack the data using unpack( ) function
   2c. display the data (i.e. usn, name, age) on screen
3. Close the file
```

**Function search( ):** This function searches for a record in the file based on a given usn using sequential search.

```
1. Read the usn to be searched
2. Open the file in read mode
3. Repeat the following,
   3a. read a record
   3b. unpack the read record
   3c. compare the usn field with the usn entered in step 1.
   3d. if both the usn are same then display the record;
       close the file and terminate.
   3e. otherwise, repeat step 3.
4. Display “record not found”
5. Close the file.
```

**Function modify( ):** this function searches a record in the file based on the given usn, if found then modify/change the data of that student otherwise display a message.

```
1. Read the usn to be searched
2. Open the file in read/write  mode
   fstream f;
     f.open("student",ios::in|ios::out);
3. Repeat the following,
   3a. read a record
   3b. unpack the read record
   3c. compare the usn field with the usn entered in step 1.
   3d. if same, change the name and age;
       pack the new data with the same usn;
       write the packed data into the same position of the file;
       close the file and terminate.
   3e. otherwise, repeat step 3.
4.  Display “record not found”
5.  Close the file.
```

**Algorithm:**

```
    1.  Create a menu for selection of users choice
        student s;  //Object s
    do {
    cout<<"\n1. Add\n2. Display\n3. Search\n4. Modify\n5.
         Exit\n\nEnter choice: ";
         cin>>ch;
         switch(ch)
         {
         case 1:
             s.add();
             break;
         case 2:
             s.display();
             break;
         case 3:
             s.search();
             break;
         case 4:
             s.modify();
             break;
         case 5:
             break;
         default:
             cout<<"Wrong choice!";
         }
    }
    while(ch!=5);
```

/*************************************************************************
*

**Output –**

```
1. Add
2. Display
3. Search
4. Modify
5. Exit

Enter choice: 1
Enter USN: 42
Enter name: Abcdef
Enter age: 21
Student record added

1. Add
2. Display
3. Search
4. Modify
5. Exit

Enter choice: 2

42|Abcdef|21

1. Add
2. Display
3. Search
4. Modify
5. Exit

Enter choice: 4
Enter the key: 42

Student found
Enter new name: Kopparam
Enter new age: 20

*************************************************************************/
```

/*********************************************************************
********

**Problem Statement –**

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

*********************************************************************
*******/

**Aim:** The aim of this problem is to create a text file to store student information using fixed length record where fields of a record is delimited by '|' and each of the records will be addressed by its RRN.We need to implement a function  to access a record using its RRN to reduce the search time.

**Concepts:**
**Keywords:** RRN (relative record number)
**Relative Record Number (RRN):** It's an integer that is used to represent a fixed length record. RRN starts with 0 followed by 1,2,3,…. where RRN 0 represent 1st record, RRN 1 represent 2nd record and so on. If length of a record is N and RRN of that record is R then address of that record is calculated as R * N.

```
#define LEN 48

class student
{
   char name[25],usn[11],age[3],buf[LEN];
  void pack();
   void unpack();
public:
   void write_file();
   void read_file();
};
```

pack() and unpack() functions are same as program 2.
**Function write_file():**This function adds one record to the student file (similar to add( ) of the previous problem)

```
void student::write_file()
{
   cout<<"Enter USN: ";
   cin>>usn;
   cout<<"Enter name: ";
   cin>>name;
   cout<<"Enter age: ";
   cin>>age;
   pack();
fout.open("student",ios::app);
   fout<<buf<<endl;
   cout<<"Student record added\n";
 fout.close();
  }
```

**Function read_file():**This function reads a RRN of a record and searches for the record based on the RRN provided. If the record with the given RRN is found then it displays the desired record.

```
void student::read_file()
{
   int rrn;
   char ch;
   cout<<"Enter the RRN: ";
   cin>>rrn;
   fin.open("student");
   fin.seekg(rrn*(LEN+1),ios::beg);
   fin>>buf;
   unpack();
   cout<<"\nUSN: "<<usn<<"\nName: "<<name<<"\nAge: "<<age<<endl;
   fin.close();
 }
```

**Algorithm:**

1. Create the user menu

```
   student s;
   do
   {
     cout<<"\n1. Add a record\n2. Display a record\n3.
     Exit\n\nEnter choice: ";
     cin>>ch;
     switch(ch)
     {
     case 1:
        s.write_file();
        break;
     case 2:
        s.read_file();
        break;
     case 3:
        break;
     default:
        cout<<"Wrong choice!";
     }
   }while(ch!=3);
```

/***************************************************************************

**Output –**

```
1. Add a record
2. Display a record
3. Exit

Enter choice: 1
Enter USN: 42
Enter name: Abcdef
Enter age: 21
Student record added

1. Add a record
2. Display a record
3. Exit

Enter choice: 1
Enter USN: 46
Enter name: skc
Enter age: 20
Student record added

1. Add a record
2. Display a record
3. Exit

Enter choice: 2
Enter the RRN: 1

USN: 46
Name: skc
Age: 20

**************************************************************************/
```

```
/**************************************************************************
Problem Statement -
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.
**************************************************************************/
```

**Aim:** To create a text file with fixed length records or variable length records, an index file with key(USN) and record address where entire record can be found and to show how indexing is performed

**Concepts:**

**Key words:**

**Index table:** a table containing key and its address
**Primary index:**unique key that represents a record in a file
**key:** an entry in index table which uniquely identifies the record.

Operations performed on index file

1. Create the original empty index and data files
2. Load the index file into memory before using it

3. Rewrite the index file after using it
4. Add data records to the data file
5. Delete records from the data file
6. Update the index file to reflect the changes in the data file

Functions used in the above program

//creating the index file.

```
Student::Student ()
{
  int i; n = 0;
  fin.open ("index");
  if (!fin.fail ())
  {
     for (i = 0; !fin.eof (); i++)
     {
        fin >> buf[i];
        if (fin.fail ())
           break;
     }
     n = i;
     fin.close ();
  }
  fin.open ("student");
  fout.open ("student", ios::app);
}


//Adding record to the data file and
void Student::Add ()
{
  int i, j, pos;
  char *temp, temp1[20];
  cout << "Enter USN: ";
  cin >> usn;
  cout << "Enter name: ";
  cin >> name;
  cout << "Enter age: ";
  cin >> age;
  pack ();
  pos = fout.tellp ();
  fout << str << endl;
  fout.flush ();
  sprintf (buf[n], "%s|%d", usn, pos);
  cout << "Student record added\n";
  n++;
}
```

```
int Student::Find (char buf[][20], char *key, int n)
{

     char *usn, temp[50];
     for(int i=0;i<n;i++)
    {
       strcpy (temp, buf[i]);
       usn = strtok (temp, "|");
       if (strcmp (key, usn) == 0) return i;
     }
     return -1; //key not found
}

//Searching for a record
void Student::Search ()
{
   char key[4], temp[20], *temp1;int index, RecAddr;
   cout << "Enter the key: "; cin >> key;
   if ((index = Find (buf, key, n)) == -1)
       cout << "Record not found!\n";
   else
   {
       cout << "Student found!\n";
       strcpy (temp, buf[index]);
       temp1 = strtok (temp, "|");
       RecAddr = atoi (strtok (NULL, "\n"));
       fin.seekg (RecAddr, ios::beg);
       fin >> str;
unpack ();
       cout << "Name: " << name;
       cout << "\nUSN: " << usn;
       cout << "\nAge: " << age << endl;
   }
}

void student::del ()
{
   char key[4];
   int index;
   cout << "Enter the key: ";
   cin >> key;
   if ((index = find (buf, key, n)) == -1)
       cout << "Record not found!\n";
   else
   {
       cout << "Record deleted\n";
       buf[index][0] = '*';
   }
}
```

```
//update the index file
Student::~Student ()
{
  fout.close ();
  fout.open ("index");
  for (int i = 0; i < n; i++)
     if (buf[i][0] != '*')
         fout << buf[i] << endl;
  fout.close ();
  fin.close ();
}
```

```
/*****************************************************************************
***
Problem Statement -
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.
*****************************************************************************
**/
```

**Aim:** To create a text file with fixed length records or variable length records, an index file with primary key(USN),secondary key(name) and record address where entire student record can be found and to show how indexing is performed

**Concepts:**

**Key words:**

**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

1. Create the original empty index and data files
2. Load the index file into memory before using it
3. Rewrite the index file after using it
4. Add data records to the data file
5. Delete records from the data file
6. Update the index file to reflect the changes in the data file

Functions used the above program

```
//creating the index file.
Student::Student ()
{
  int i; n = 0;
  fin.open ("index");
  if (!fin.fail ())
  {
     for (i = 0; !fin.eof (); i++)
     {
        fin >> buf[i];
        if (fin.fail ())
           break;
     }
     n = i;
     fin.close ();
  }
  fin.open ("student");
  fout.open ("student", ios::app);
}



//Adding record to the data file

void Student::Add ()
{
  int i, j, pos;
  char *temp, temp1[20];
  cout << "Enter USN: ";
  cin >> usn;
  cout << "Enter name: ";
  cin >> name;
  cout << "Enter age: ";
  cin >> age;
  Pack ();
  pos = fout.tellp ();
  fout << str << endl;
  fout.flush ();
  sprintf (buf[n], "%s|%s|%d", name,usn,pos);//seconday index
  cout << "Student record added\n";
  n++;
}

int Student::Find (char buf[][20], char *key, int n)
{

    char *name,temp[50];
    for(int i=0;i<n;i++)
   {
     strcpy (temp, buf[i]);
     name = strtok (temp, "|");
     if (strcmp (key, name) == 0) return i;
    }
    return -1; //key not found
```

```
}



//Searching for a record
void Student::Search ()
{
  char key[4], temp[20], *tempusn,*tempname;
int index, RecAddr;
  cout << "Enter the key: "; cin >> key;
  if ((index = Find (buf, key, n)) == -1)
      cout << "Record not found!\n";
  else
  {
      cout << "Student found!\n";
      strcpy (temp, buf[index]);
      tempusn = strtok (temp, "|");
      tempname = strtok (NULL, "|");
      RecAddr = atoi (strtok (NULL, "\n"));
      fin.seekg (RecAddr, ios::beg);  //move the file pointer to the data record in data file
      fin >> str;                            // Read the record
      Unpack ();
      cout << "Name: " << name;     /* display the individual fields */
      cout << "\nUSN: " << usn;
      cout << "\nAge: " << age << endl;
  }
}

// delete function

void student::del ()
{
  char key[4];
  int index;
  cout << "Enter the key: ";
  cin >> key;
  if ((index = find (buf, key, n)) == -1)
      cout << "Record not found!\n";
  else
  {
      cout << "Record deleted\n";
      buf[index][0] = '*';          // mark the buffer with * for the deleted record
  }
}


//updating the index table
Student::~Student ()
{
  fout.close ();
  fout.open ("index");
  for (int i = 0; i < n; i++)
      if (buf[i][0] != '*')  //copy the records to the index file not marked with *
          fout << buf[i] << endl;
  fout.close ();
  fin.close ();
}
```

/***********************************************************************
**Problem Statement –**
7. Write a C++ 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.

************************************************************************
****/

**AIM:** The aim of this program is to read two lists of names from standard input device like keyboard and using consequential match based on a single loop compare both the list of names and write the names to a file which are common to both the files and print them on standard output device like monitor.

**Concepts -**

**Keywords:** Co-sequential match

**Co-sequential Match**: Coordinated processing of two or more sequential lists to produce a single list.

We have at least two lists and they contain names in sorted order. From these two list we find out the common names and write them to a new file.
It contains the match procedure as given below:

1. Initializing: we need to arrange the data in sorted order such a way that the procedure gets going properly.
2. Getting and accessing the next list item: we need simple methods that support getting the next list element and accessing it.
3. Synchronizing: we have to make sure that match will never be missed.
4. Handling end of file conditions: when to get to the end of either of Lists we need to halt the program.
5. Recognizing errors: when an error occurs in the data, we need to detect it and take some action.

**Algorithms:**

1. Start

2. Take two lists l1, l2 such that data in it is in sorted order.

3. Open these files f1.open("l1");
```
                              f2.open("l2");
```
Open the output file, fout.open("out")

4. Read from first file and store the data in buf1
```
f1>>buf1;
if(f1.fail()) // if read error occurs
  {
     cout<<"List 1 empty\n";
     return 0;
  }
```

5. Read from second file, store the data in buf2; check for read error.

6. Repeat the following until comparison of data between the two files are over
   i. If values are equal then we will write these values on the new file, read the next data from both files
   ii. Or if buf1 has lesser value than buf2, read the next data from first file
   iii. else read next data from second file.

```
while(1)
  {
     cmp_val=strcmp(buf1,buf2);
     if(cmp_val == 0)
     {
        fout<<buf1<<endl;
        f1>>buf1;
        if(f1.fail())
           break;
        f2>>buf2;
        if(f2.fail())
           break;
     }
else if(cmp_val < 0)
     {
        f1>>buf1;
        if(f1.fail())
           break;
      }

else
     {
        f2>>buf2;
        if(f2.fail())
        break;
    }
}
```

9.Stop

/***************************************************************************
******

**Output -**

**//Displays on console**

```
abigith
kashyap
neha
nikita
```

**Contents of l1 – [1st file]**

```
abigith
deepa
kashyap
kfc
neha
nikita
```

**Contents of l2– [2nd file]**

```
abigith
kashyap
keerti
neha
nikita
upasana
```

**Contents of out– [output file]**

```
abigith
kashyap
neha
nikita
```

****************************************************************************
*****/

/***************************************************************************
******

**Problem Statement –**

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

*************************************************************************
*****/

**Aim:** The aim of this program is to read K lists of names from standard input in sorted order in each list and merge K number of lists in to one sequentially ordered list using K-way Merge Algorithm and print merged list on standard output device.

**Concepts -**

**Keywords:** K-way Merge Algorithm

**K-way Merge Algorithm:** It is to merge K input lists to create a single sequentially ordered list. It can be viewed as a process of deciding which two input items has the minimum value, outputting that item, then moving ahead in the list from which that item is taken. In this we take all k lists in sorted order.

It requires calling a minimum index function to find the index of item with the minimum collating sequence value and an inner loop that finds all lists that are using that item.In this we find the minimum and testing to see that in which list the item occurs and which files therefore need to be read.

It works nicely if k value is less than or equal to 8

**Algorithm:**

1. Start.
2. We create 8 files with names in sorted order in each file.
3. Assign variables for processing of 8 files where open_files=8 (number of files currently being opened) and more_names is a variable that checks whether all the names of the file has been read or not. Ex., if more_names[i] = 1, some names of the ith file has not been processed yet and more_names[i] = 0, otherwise.

```
    intopen_files=8;
int more_names[8]={1,1,1,1,1,1,1,1};
```

4. It will display the details of all the files and will open all of them one by one and will read the first data of all the files one by one.

```
for(int i=0;i<8;i++)
{
    sprintf(file_name,"file%d",i);
    fin[i].open(file_name);
    fin[i]>>buf[i];
    }
```

5. When opening the files we have to find the minimum value among first data of all the files

```
while(open_files)  //repeat the following until all the files are get closed
{
        min=find_min(buf,more_names); //find the minimum
        if(strcmp(buf[min],prev_name)) //if two names are same remove duplicate in output file
    {
       strcpy(prev_name,buf[min]);
        fout<<buf[min]<<endl;       // write the minimum in the output file
        cout<<buf[min]<<endl;
        fin[min]>>buf[min];          //read the next data from the file where minimum data was
                                       present
        if(fin[min].fail())          // reach end of a file
        {
          more_names[min]=0;
          open_files--;
        }
        }
        else   // no duplicate entries
        {
        fin[min]>>buf[min];
        if(fin[min].fail())
        {
          more_names[min]=0;
          open_files--;
        }
    }
```

In this code we will calculate minimum of two values each from the lists. In this we assume first data of first list as min and will compare it with the first data of rest of all the lists and if any other value is lesser than the min value then min value will be the other one.

```
int find_min(char buf[][25],int more_names[])
{
  int min;
  for(min=0;min<8;min++)
     if(more_names[min])
      break;
  for(int i=min+1;i<8;i++)
     if(strcmp(buf[min],buf[i]) > 0 && more_names[i])
        min=i;
return min;
}
```

/*************************************************************************
******
**Output -**

```
kashyap
keerti
kfc
kp
neha
niki
nikita
nivedita
```

**Contents of file0 -**
```
kfc
niki
nivedita
```

**Contents of file1 -**
```
kashyap
kfc
kp
```

**Contents of file2 -**
```
kashyap
neha
nikita
```

**Contents of file3 -**
```
keerti
kfc
niki
```

**Contents of file4 -**
```
kfc
niki
nivedita
```

**Contents of file5 -**
```
kashyap
kfc
kp
```

**Contents of file6 -**
```
kashyap
neha
nikita
```

**Contents of file7 -**
keerti
kfc
niki

**************************************************************************
*****/