FILE STRUCTURES LABORATORY MANUAL 10ISL67

MAHARAJA INSTITUTE OF TECHNOLOGY
Behind K R Mills, Belawadi Village, MYSORE.

VI SEMESTER

DEPT. OF INFORMATION SCIENCE & ENGINEERING

FILE STRUCTURES LABORATORY

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.

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

3. Write a C++ program to read and write and student objects with variable-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 key for a file of student objects. Implement add(),search(),delete() using the index.

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.

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.

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

9. Write a C++ program to implement B-Tree for a given set of integers and its operations insert ( ) and search ( ). Display the tree.

10. Write a C++ program to implement B+ tree for a given set of integers and its operations insert ( ), and search ( ). Display the tree.

11. Write a C++ program to store and retrieve student data from file using hashing. Use any collision resolution technique

12. Write a C++ program to reclaim the free space resulting from the deletion of records using linked lists.

File Structures Laboratory

Subject Code: 06ISL67 **I.A. Marks: 25**
Hours/Week: 03 **Exam Hours: 03**
Total Hours: 42 **Exam Marks: 50**

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.**

I/O redirection

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

- In both DOS and UNIX, the standard output of a program can be redirected to a file with the > symbol.
- In both DOS and UNIX, the standard input of a program can be redirected to a file with the < symbol.
- The notations for input and output redirection on the command line in Unix are

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

- Example:

```
list.exe > myfile
```

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

pipe

- Piping: using the output of one program as input to another program. A connection between standard output of one process and standard input of a second process.
- In both DOS and UNIX, the standard output of one program can be piped (connected) to the standard input of another program with the | symbol.
- Example:

```
program1 | program2
```

- Output of program1 is used as input for program2

File I/O:

perform output and input of characters to or from files

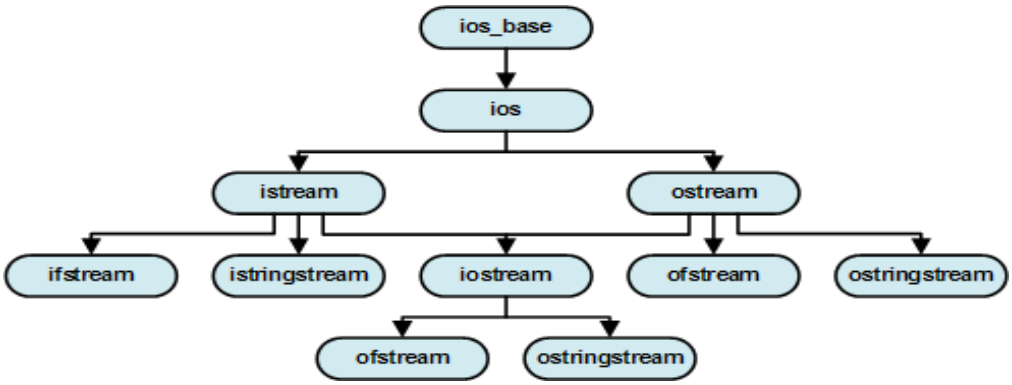
Standerd I/O:

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

fstream

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

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



Function to open a file:

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

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

```
open (filename, mode);
```

Where `filename` is a null-terminated character sequence of type `const char *` (the same type that string

literals have) representing the name of the file to be opened, and `mode` is an optional parameter with a combination of the following flags:

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

Function to Closing the Files

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

- **Prototypes:**

  int close (int Handle);

- **Example:**

  close (Input);

Getline Function:

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

- **prototype**

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

Program 1

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

class std_file
{
private:
      char name[10][20];
      char input[20],output[20],str[20];
public:
      void std_io();
      void file_io();
};

void std_file::std_io(){
int n,i;
cout<<"enter the number of names to read "<<endl;
           cin>>n;
           cout<<"enter the names"<<endl;
           for(i=0;i<n;i++)
           gets(name[i]);
           cout<<"the reversed names are"<<endl;
           for(i=0;i<n;i++)
             cout<<strrev(name[i])<<endl;

}
void std_file::file_io(){
      fstream ifile,ofile;
      cout<<"enter the filename which contain list of names"<<endl;
      cin>>input;
      ifile.open(input,ios::in);
      if(!ifile)
      {
            cout<<"file doesnot exist";
            exit(0);
            getch();
      }
      cout<<"enter the filename to store names in reverse order"<<endl;
      cin>>output;
      ofile.open(output,ios::out);
      while(!ifile.eof())
      {
            ifile.getline(str,20);
            ofile<<strrev(str)<<endl; //to reverse string characters
      }
}
```

```
void main()
{
      int i,num,len;
      std_file s;
      clrscr();
      for(;;)
      {
      cout<<"1:file i/o\n2:standard i/o\noany other to exit\n";
      cin>>num;
      switch(num){
      case 1:
            s.std_io();
            break;

      case 2:
            s.file_io();
            break;
      default:
            exit(0);
                  }
     }
}
```

Output 1:

1: file i/o
2: standard i/o
0 : any other to exit
1
enter the number of names to read
2
Enter the names
keerthana
madhu
The reversed names are
anahtreek
uhdam

1: file i/o
2: standard i/o
0: any other to exit
2
Enter the filename which contain list of names
student.txt
Enter the filename to store names in reverse order
studentr.txt

1:file i/o
2:standard i/o

any other to exit
0
c:\tc>type student.txt
ragu
navya

c:\tc>type studentr.dat
ugar
ayvan

Output 2:

1:file i/o
2:standard i/o
any other to exit
2
enter the filename which contain list of names
std.txt
file does not exist

1:file i/o
2:standard i/o
any other to exit
0

Output 3: using I/O redirection and pipes (Run the program in Command prompt)

I/O redirection : Redirect the output from *stdout* to a file aaa.txt
Syntax : program1 >filename
Exp:
c:\tc>fslab1 > aaa.txt
1
3
keerthana
madhu
bavana
1:file i/o
2:standard i/o
any other to exit
0
c:\tc>type aaa.txt
1:file i/o
2:standard i/o

any other to exit
enter the number of names to read
enter the names
the reversed names are
anahtreek
uhdam
anavab
1:file i/o
2:standard i/o
any other to exit

Pipes :take any *stdout* output from program 1 and use it in place of any *stdin* input to program2.
Syntax : program1 | program 2

c:\tc>type student.txt | sort
bavana
keerthana
madhu

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

Fixed length record

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

Record 1	Record 2	Record 3	Record 4	Record 5

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

Delimited Variable Length Fields

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

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

Pack():

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

Unpack():

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

File_structure2.cpp

```
#include<iostream.h>
#include<fstream.h>
#include<process.h>
#include<string.h>
#include<conio.h>
class student
{
      private:
            char buf[45],name[10],sem[10],branch[10];
      public:
            void read()
            {
                  cout<<"Name: "<<endl;
                  cin>>name;
                  cout<<"Semester: "<<endl;
                  cin>>sem;
                  cout<<"Branch: "<<endl;
                  cin>>branch;
            }

            void pack(fstream &ofile)
            {
                  read();
                  strcpy(buf,"");
                  strcat(buf,name);
                  strcat(buf,"|");
                  strcat(buf,sem);
                  strcat(buf,"|");
                  strcat(buf,branch);
                  strcat(buf,"|");
                  while(strlen(buf)<45)
                        strcat(buf,"!");
                  strcat(buf,"\n");
                  ofile.write(buf,strlen(buf));
            }

            void  unpack(fstream &ifile)
            {
                  char extra[45];
                  while(!ifile.eof())
                  {
                        ifile.getline(name,10,'|');
                        ifile.getline(sem,10,'|');
                        ifile.getline(branch,10,'|');
```

```
                ifile.getline(extra,45,'\n');
                cout<<name<<"\t"<<sem<<"\t"<<branch<<"\n";
            }
        }

        int search(fstream &ifile,char key[])
        {
            char extra[45];
            while(!ifile.eof())
            {
                ifile.getline(name,10,'|');
                ifile.getline(sem,10,'|');
                ifile.getline(branch,10,'|');
                ifile.getline(extra,45,'\n');
                if(strcmp(name,key)==0)
                {
                    cout<<"Record found and details are:"<<endl;
                    cout<<"Name: "<<name<<endl;
                    cout<<"Semester: "<<sem<<endl;
                    cout<<"Branch: "<<branch<<endl;
                    return 1;
                }
            }
            return 0;
        }

        void modify(fstream &iofile,char key[])
        {
            if(search(iofile,key))
            {
                cout<<"Record found,enter modification details:"<<endl;
                iofile.seekp(-47,ios::cur);
                pack(iofile);
            }
            else
                cout<<"Sorry!No such record\n";
        }
};

void main()
{
    int n,i,ch;
    student stu;
    fstream ofile;
    ofile.open("student.txt",ios::trunc|ios::app);
```

```
      ofile.close();
      clrscr();
      for(;;)
      {
            clrscr();
            cout<<"1. Insert\n2.   Display  all\n3.   Search\n4.   Modify\n5.
Exit\n";
            cout<<"Enter your choice"<<endl;
            cin>>ch;
            switch(ch)
            {
                  case 1:    fstream ofile;
                        ofile.open("student.txt",ios::out|ios::app);
                        cout<<"Enter the no. of students"<<endl;
                        cin>>n;
                        for(i=0;i<n;i++)
                        {
                              stu.pack(ofile);
                        }
                        ofile.close();
                        break;

                  case 2:    fstream infile;
                        infile.open("student.txt",ios::in);
                        stu.unpack(infile);
                        getch();
                        infile.close();
                        break;

                  case 3: cout<<"Enter the record name to be searched"<<endl;
                        char key[10];
                        cin>>key;
                        fstream ifile;
                        ifile.open("student.txt",ios::in);
                        if(stu.search(ifile,key)==0)
                              cout<<"record not found\n";
                        getch();
                        ifile.close();
                        break;

                  case 4:    fstream iofile;
                        iofile.open("student.txt",ios::in|ios::out);
                        cout<<"Enter the record name to be modified"<<endl;
                        cin>>key;
                        stu.modify(iofile,key);
```

```
                    getch();
                    iofile.close();
                    break;
               default: exit(0);
          }
     }
}
```

Output :

```
1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:1
Enter the number of students:2

Enter the student name = ajay
Enter the sem = 6
Enter the branch = ise

Enter the student name = rahul
Enter the sem = 6
Enter the branch = cse


1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:2
Name              Sem         Branch
ajay              6           ise||||||||||
rahul             6           cse||||||||||



1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:4
Enter the record name you want to search = rahul
Record found
rahul       6           cse||||||||||



1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:3
Enter the record name you want to modify:rahul
record found and details are:

rahul       6           cse||||||||||

enter modification details
```

```
Enter the student name =navya
Enter the sem = 6
Enter the branch = ise


1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:2
Name              Sem         Branch
ajay              6           ise||||||||||
Navya             6           ise||||||||||


1:write to file 2:display the file 3:modify the file 4:search 5.exit

Enter the choice:4
Enter the record name you want to search:keerthi
Record not found
```

3. Write a C++ 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.

File_structure3.cpp

```
#include<iostream.h>
#include<fstream.h>
#include<process.h>
#include<string.h>
#include<conio.h>
class student
{
      private: char buf[45],name[10],sem[10],branch[10];
      int pos;
      public:
      void read()
      {
            cout<<"name:"<<endl;
            cin>>name;
            cout<<"semester:"<<endl;
            cin>>sem;
            cout<<"branch:"<<endl;
            cin>>branch;
      }

      void pack(fstream &ofile)
       {
            read();
            strcpy(buf,"");
            strcat(buf,name);
            strcat(buf,"|");
            strcat(buf,sem);
            strcat(buf,"|");
            strcat(buf,branch);
            strcat(buf,"|");
            strcat(buf,"\n");
            ofile.write(buf,strlen(buf));
      }

      void unpack(fstream &ifile)
      {
            char extra[45];
            while(!ifile.eof())
            {
                  ifile.getline(name,10,'|');
                  ifile.getline(sem,10,'|');
                  ifile.getline(branch,10,'|');
                  ifile.getline(extra,45,'\n');
                  cout<<name<<"\t";
                  cout<<sem<<"\t";
                  cout<<branch<<"\n";
            }
      }
```

```
int search(fstream &ifile,char key[])
{
      char extra[45];
      while(!ifile.eof())
      {
            ifile.getline(name,10,'|');
            ifile.getline(sem,10,'|');
            ifile.getline(branch,10,'|');
            ifile.getline(extra,45,'\n');

             if(strcmp(name,key)==0)
             {
                   cout<<" "<<"record found and details are:"<<endl;
                   cout<<" "<<"name"<<name<<endl;
                  cout<<" "<<"semester"<<sem<<endl;
                   cout<<" "<<"branch"<<branch<<endl;
                   return 1;
             }

      }
       return 0;
}

void modify(fstream &ifile,char key[])
{
      student s[10];
      char extra[50];
      int i=0;

       while(!ifile.eof())
       {
            ifile.getline(s[i].name,10,'|');
            ifile.getline(s[i].sem,10,'|');
            ifile.getline(s[i].branch,10,'|');
            ifile.getline(extra,45,'\n');
            i++;
       }
      ifile.close();
      int flag=0;
      for(int j=0;j<i;j++)
      {
            if(strcmp(key,s[j].name)==0)
            {
                  flag=1;
                  cout<<"record found details are:"<<endl;
                  cout<<s[j].name<<endl;
                  cout<<s[j].sem<<endl;
                  cout<<s[j].branch<<endl;
```

```
                        cout<<"enter the modification details"<<endl;
                        cout<<"enetr the name"<<endl;
                        cin>>s[j].name;
                        cout<<"enter the sem;"<<endl;
                        cin>>s[j].sem;
                        cout<<"enter the branch"<<endl;
                        cin>>s[j].branch;
                  }
            }
            if(flag==0)
            {
                  cout<<"Record not found\n";
                  return;
            }
            ifile.open("student.txt",ios::trunc|ios::app);
            for(int k=0;k<i;k++)
            {
                  strcpy(buf,"");
                  strcat(buf,s[k].name);
                  strcat(buf,"|");
                  strcat(buf,s[k].sem);
                  strcat(buf,"|");
                  strcat(buf,s[k].branch);
                  strcat(buf,"|");
                  strcat(buf,"\n");
                  ifile.write(buf,strlen(buf));
            }
      }
};
void main()
{
      int n,i,ch;
      char key[10];
      student stu;
      fstream ifile,ofile;
      ofile.open("student.txt",ios::trunc|ios::app);
      ofile.close();
      for(;;)
      {
            clrscr();
            cout<<"1.insert\n 2.display\n 3.search\n 4.modify\n 5.exit\n";
            cout<<"enter your choice"<<endl;
            cin>>ch;
            switch(ch)
            {
                  case 1: fstream ofile;
                        ofile.open("student.txt",ios::out|ios::app);
                        cout<<"enter the no of students";
                        cin>>n;
```

```
                        for(i=0;i<n;i++)
                        {
                              stu.pack(ofile);
                        }
                        ofile.close();
                        break;

                  case 2: fstream infile;
                         infile.open("student.txt",ios::in);
                         stu.unpack(infile);
                         getch();
                         infile.close();
                         break;

                  case 3:cout<<"enter the record name to be searched"<<endl;
                         cin>>key;
                         fstream ifile;
                         ifile.open("student.txt",ios::in);
                         if(stu.search(ifile,key)==0)
                         cout<<"record not found\n";
                         getch();
                         ifile.close();
                         break;

                  case 4: fstream iofile;
                         iofile.open("student.txt",ios::in|ios::out);
                         cout<<"enter the record name to be modified\n"<<endl;
                         cin>>key;
                         stu.modify(iofile,key);
                         getch();
                         iofile.close();
                         break;
                         default:exit(0);
                   }
            }
      }
```

Output:

```
1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:1
Enter the number of students:2
Enter the student name = ajay
Enter the sem = 6
Enter the branch = ise

Enter the student name = rahul
Enter the sem = 6
Enter the branch = cse

1:write to file 2:display the file 3:modify the file 4:search 5.exit
```

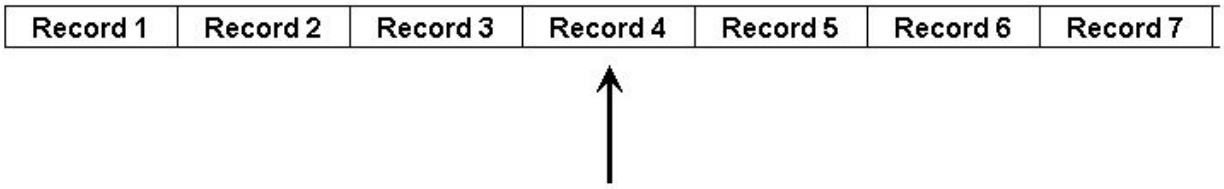
```
Enter the choice:2
Name              Sem         Branch
ajay              6           ise
rahul             6           cse

1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:4
Enter the record name you want to search = rahul
Record found
rahul       6           cse

1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:3
Enter the record name you want to modify:rahul
record found and details are:
rahul       6           cse

enter modification details
Enter the student name =navya
Enter the sem = 6
Enter the branch = ise
1:write to file 2:display the file 3:modify the file 4:search 5.exit
Enter the choice:2
Name              Sem         Branch
ajay              6           ise
Navya             6           ise

1:write to file 2:display the file 3:modify the file 4:search 5.exit

Enter the choice:4
Enter the record name you want to search:keerthi
Record not found
```

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.

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
    - RRN: relative record number(starts fron 0)
    - RecLen: size of fixed lenth record

Direct Access



File_structure4.ccp

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

class student
{
      private:
            char buf[40],name[10],sem[10],branch[10],extra[40];
      public:
            void read()
            {
                  cout<<"Name: "<<endl;
                  cin>>name;
                  cout<<"Semester: "<<endl;
                  cin>>sem;
                  cout<<"Branch: "<<endl;
                  cin>>branch;
            }

            void insert(fstream &ofile,char rrn[])
            {
                  read();
                  strcpy(buf,"");
                  strcat(buf,rrn);
                  strcat(buf,"|");
                  strcat(buf,name);
                  strcat(buf,"|");
                  strcat(buf,sem);
                  strcat(buf,"|");
                  strcat(buf,branch);
                  strcat(buf,"|");
                  strcat(buf,"\n");
                  ofile.write(buf,strlen(buf));
            }

            int search(fstream &ifile,char key[])
            {
                  char rrn[10];
                  while(!ifile.eof())
                  {
                        ifile.getline(rrn,10,'|');
                        ifile.getline(name,10,'|');
                        ifile.getline(sem,10,'|');
                        ifile.getline(branch,10,'|');
                        ifile.getline(extra,40,'\n');
```

```
                        if(strcmp(key,rrn)==0)
                        {
                              cout<<"Record found and details are:"<<endl;
                              cout<<"Name: "<<name;
                              cout<<"Semester: "<<sem;
                              cout<<"Branch: "<<branch;
                              return 1;
                        }
                  }
                  return 0;
            }
};

void main()
{
      int n,i,ch,k=0;
      char key[10];
      student stu;
      fstream ofile;
      ofile.open("student2.txt",ios::trunc|ios::app);
      ofile.close();
      clrscr();
      for(;;)
      {
            cout<<"1.Insert\n2.Search\n3.Exit\n";
            cout<<"Enter your choice: ";
            cin>>ch;
            switch(ch)
            {
                  case 1: fstream ofile;
                        ofile.open("student2.txt",ios::out|ios::app);
                        cout<<"Enter the no. of students: ";
                        cin>>n;
                        for(i=0;i<n;i++)
                        {
                              itoa(++k,key,10);
                              stu.insert(ofile,key);
                        }
                        ofile.close();
                        break;
                  case 2: cout<<"Enter the RRN to search: ";
                        cin>>key;
                        fstream ifile;
                        ifile.open("student.txt",ios::in);
                        if(stu.search(ifile,key)==0)
                        cout<<"Record not found\n";
                        ifile.close();
                        break;
                  default:exit(0);
```

```
            }
      }
}
```

Output:

```
1.Insert
2.Search
3.Exit
Enetr your choice:1
Enter the no. of students:2
name = ajay
sem = 6
branch = ise

name = rahul
sem = 6
branch = cse

1.Insert
2.Search
3.Exit
Enetr your choice:2
Enter the RRN to search:1

Record found and details are:"<<
rahul             6           cse
1.Insert
2.Search
3.Exit
Enetr your choice:2
Enter the RRN to search:5
Record not found
```

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.

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

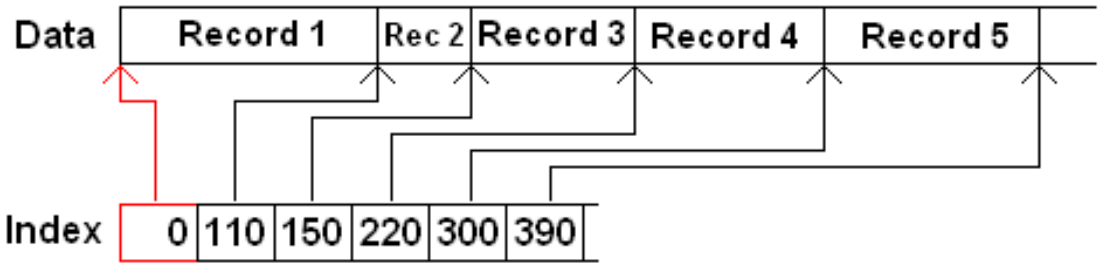
The part of an index which contains information to locate records.

- 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.



File_structure5.ccp

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

int n=0,index=0;

class student
{
      public: char name[20],usn[20],branch[5];
            int sem;

            void insert(fstream &f1,fstream &f2)
            {
                  cout<<"Enter Name: ";
                  cin>>name;
                  cout<<"Enter USN: ";
                  cin>>usn;
                  cout<<"Enter Sem: ";
                  cin>>sem;
                  cout<<"Enter Branch: ";
                  cin>>branch;
                  write(f1,f2);
            }

            void write(fstream &f1,fstream &f2)
            {
                  f1<<++index<<"\t"<<usn<<"\n";
                  f2<<name<<"\t"<<usn<<"\t"<<sem<<"\t"<<branch<<"\n";
            }

            void display(fstream &f2)
            {
                  f2>>name>>usn>>sem>>branch;
                  cout<<name<<"\t"<<usn<<"\t"<<sem<<"\t"<<branch<<"\n";
            }

            int search(fstream &f1,char key[20])
            {
                  int i,x;
                  for(i=1;i<=n;i++)
                  {
                        f1>>x>>usn;
                        if(strcmp(usn,key)==0)
                              return i;
                  }
                  cout<<"Record not found\n";
```

```
                return 0;
            }

            int remove(fstream &f1,char key[20])
            {
                int i;
                i=search(f1,key);
                return i;
            }
};

void main()
{
      fstream f1,f2;
      student s[20],p;
      int ch,k=0,i;
      clrscr();
      f1.open("m1.txt",ios::trunc);
      f2.open("mn1.txt",ios::trunc);
      f1.close();
      f2.close();
      for(;;)
      {
            cout<<"1.Insert 2.Display 3.Search 4.Delete 5.Exit\n";
            cout<<"Enter choice: ";
            cin>>ch;
            switch(ch)
            {
                  case 1: f1.open("m1.txt",ios::app);
                        f2.open("mn1.txt",ios::app);
                        cout<<"Enter no. of students: ";
                        cin>>k;
                        n=n+k;
                        for(int i=1;i<=k;i++)
                              s[i].insert(f1,f2);
                        f1.close();
                        f2.close();
                        break;
                  case 2: f2.open("mn1.txt",ios::in);
                        for(i=1;i<=n;i++)
                              s[i].display(f2);
                        f2.close();
                        break;
                  case 3: char usn[20];
                        cout<<"Enter USN to search: ";
                        cin>>usn;
                        f1.open("m1.txt",ios::in);
                        f2.open("mn1.txt",ios::in);
                        int j=p.search(f1,usn);
```

```
                        if(j!=0)
                        {
                              cout<<"Record found & Details are\n";
                              cout<<"Name="<<s[j].name<<"\n"<<"USN="<<s[j].usn<<"\n"
                              <<"Sem="<<s[j].sem<<"\n"<<"Branch="<<s[j].branch<<"\n";
                        }
                        f1.close();
                        f2.close();
                        break;
                  case 4: f1.open("m1.txt",ios::in);
                        f2.open("mn1.txt",ios::in);
                        cout<<"Enter USN to delete: ";
                        cin>>usn;
                        j=p.remove(f1,usn);
                        if(j!=0)
                        {
                              for(i=j;i<n;i++)
                                    s[i]=s[i+1];
                              cout<<"Deletion successfull\n";
                        }
                        n--;
                        index--;
                        f1.close();
                        f2.close();
                        f1.open("m1.txt",ios::trunc|ios::app);
                        f2.open("mn1.txt",ios::trunc|ios::app);
                        index=0;
                        for(i=1;i<=n;i++)
                              s[i].write(f1,f2);
                        f1.close();
                        f2.close();
                        break;
                  default:exit(0);
            }
      }
}
```

Output :

```
1.Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice : 1

Enter the no. of students :2
Enter the details:
Name: ajay
USN: 1vk07is002
Sem: 6
Branch: ise

Name: rahul
```

```
USN: 1vk07cs045
Sem: 6
Branch: cse

1.Insert 2.Display 3.Search 4.Delete 5.Exit

Enter u'r choice: 2
ajay        1vk07is002        6           ise
rahul       1vk07cs045        6           cse


1. Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice :3
Enter USN to search:
1vk07is002
ajay              1vk07is002        6           ise

1. Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice: 4

Enter USN whose record is to be deleted: 1vk07cs045
Deletion succesfull

1. Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice: 5
```

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.

File_structure6.cpp

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

int n=0,index=0;

class student
{
      public: char name[20],usn[20],branch[5];
            int sem;

            void insert(fstream &f1,fstream &f2)
            {
                  cout<<"Enter Name: ";
                  cin>>name;
                  cout<<"Enter USN: ";
                  cin>>usn;
                  cout<<"Enter Sem: ";
                  cin>>sem;
                  cout<<"Enter Branch: ";
                  cin>>branch;
                  write(f1,f2);
            }

            void write(fstream &f1,fstream &f2)
            {
                  f1<<++index<<"\t"<<name<<"\n";
                  f2<<name<<"\t"<<usn<<"\t"<<sem<<"\t"<<branch<<"\n";
            }

            void display(fstream &f2)
            {
                  f2>>name>>usn>>sem>>branch;
                  cout<<name<<"\t"<<usn<<"\t"<<sem<<"\t"<<branch<<"\n";
            }

            int search(fstream &f1,char key[20])
            {
                  int i,x;
                  for(i=1;i<=n;i++)
                  {
                        f1>>x>>name;
```

```
                        if(strcmp(name,key)==0)
                              return i;
                  }
                  cout<<"Record not found\n";
                  return 0;
            }

            int remove(fstream &f1,char key[20])
            {
                  int i;
                  i=search(f1,key);
                  return i;
            }
};

void main()
{
      fstream f1,f2;
      student s[20],p;
      int ch,k=0,i;
      clrscr();
      f1.open("m.txt",ios::trunc);
      f2.open("mn.txt",ios::trunc);
      f1.close();
      f2.close();
      for(;;)
      {
            cout<<"1.Insert 2.Display 3.Search 4.Delete 5.Exit\n";
            cout<<"Enter choice: ";
            cin>>ch;
            switch(ch)
            {
                  case 1: f1.open("m.txt",ios::app);
                        f2.open("mn.txt",ios::app);
                        cout<<"Enter no. of students: ";
                        cin>>k;
                        n=n+k;
                        for(int i=1;i<=k;i++)
                              s[i].insert(f1,f2);
                        f1.close();
                        f2.close();
                        break;
                  case 2: f2.open("mn.txt",ios::in);
                        for(i=1;i<=n;i++)
                              s[i].display(f2);
                        f2.close();
                        break;
                  case 3: char name[20];
                        cout<<"Enter name to search: ";
```

```
                        cin>>name;
                        f1.open("m.txt",ios::in);
                        f2.open("mn.txt",ios::in);
                        int j=p.search(f1,name);
                        if(j!=0)
                        {
                              cout<<"Record found & Details are\n";
                              cout<<"Name="<<s[j].name<<"\n"<<"USN="<<s[j].usn<<"\n"
                              <<"Sem="<<s[j].sem<<"\n"<<"Branch="<<s[j].branch<<"\n";
                        }
                        f1.close();
                        f2.close();
                        break;
                  case 4: f1.open("m.txt",ios::in);
                        f2.open("mn.txt",ios::in);
                        cout<<"Enter name to delete: ";
                        cin>>name;
                        j=p.remove(f1,name);
                        if(j!=0)
                        {
                              for(i=j;i<n;i++)
                                    s[i]=s[i+1];
                              cout<<"Deletion successfull\n";
                        }
                        n--;
                        index--;
                        f1.close();
                        f2.close();
                        f1.open("m.txt",ios::trunc|ios::app);
                        f2.open("mn.txt",ios::trunc|ios::app);
                        index=0;
                        for(i=1;i<=n;i++)
                              s[i].write(f1,f2);
                        f1.close();
                        f2.close();
                        break;
                  default:exit(0);
            }
      }
}
```

Output :

```
1.Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice : 1

Enter the no. of students :2
Enter the details:
Name: ajay
USN: 1vk07is002
```

```
Sem: 6
Branch: ise

Name: rahul
USN: 1vk07cs045
Sem: 6
Branch: cse

1.Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice: 2

ajay        1vk07is002        6           ise
rahul       1vk07cs045        6           cse


1. Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice :3
Enter USN to search:
1vk07is002
ajay              1vk07is002        6           ise


1. Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice: 4

Enter name whose record is to be deleted: 1vk07cs045
Deletion succesfull


1. Insert 2.Display 3.Search 4.Delete 5.Exit
Enter u'r choice: 5
```

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.

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.)

File_structure7.cpp

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

void writeLists()
{
      fstream out1,out2;
      int i,m,n;
      char name[20];
      out1.open("file1.txt",ios::out);
      out2.open("file2.txt",ios::out);
      if( (!out1) || (!out2) )
      {
            cout<<"Unable to open one of the list files\n";
            getch();
            exit(0);
      }
      cout<<"Enter the number of names you want to enter in file1 ";
      cin>>m;
      cout<<"\nEnter the names in assending order \n";
      for(i=0;i<m;i++)
      {
            cin>>name;
            out1<<name;
            out1<<'\n';
      }

      cout<<"Enter the number of names you want to enter in file2 ";
      cin>>n;
      cout<<"\nEnter the names in assending order \n";
      for(i=0;i<n;i++)
      {
            cin>>name;
            out2<<name;
            out2<<'\n';
      }
      out1.close();
      out2.close();
}
void main()
{
      char list1[100][20],list2[100][20];
      int i,j,m,n;
      clrscr();
      fstream out1,out2,out3;
```

```
	writeLists();
	out1.open("file1.txt",ios::in);
	out2.open("file2.txt",ios::in);
	out3.open("file3.txt",ios::out);
	if((!out3)||(!out1)||(!out2))
	{
		cout<<"Unable to open one of the file";
		getch();
		exit(0);
	}
	clrscr();
	m=0;
	n=0;
	while(!out1.eof())
	{
		out1.getline(list1[m],20,'\n');
		cout<<list1[m]<<"\t";
		m++;
	}
	cout<<endl;
	while(!out2.eof())
	{
		out2.getline(list2[n],20,'\n');
		cout<<list2[n]<<"\t";
		n++;
	}
	m--;
	n--;
	i=0;
	j=0;
	cout<<"\nElements common to both files are\n";
	while(i<m&&j<n)
	{
		if(strcmp(list1[i],list2[j])==0)
		{
			out3<<list1[i];
			cout<<list1[i]<<"\n";
			out3<<'\n';
			i++;
			j++;
		}
		else if(strcmp(list1[i],list2[j])<0)
			i++;
		else
		j++;
	}
	getch();
}
```

Output :

```
Enter no. of names you want to enter in file1 : 3
Enter the names in ascending order
cse
ise
tc

Enter no. of names you want to enter in file1 : 2
Enter the names in ascending order
ec
ise

cseisetcecise

Elements common to both files are:
 Ise
```

8. Write a C++ program to read k Lists of names and merge them using kway 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.

File_structure8.cpp

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

class record
{
      public:
      char name[20];
      char usn[20];
}rec[20];

fstream file[8];
int no;
char fname[8][8]={"1.txt","2.txt","3.txt","4.txt","5.txt","6.txt","7.txt","8.txt"};

void merge_file(char* file1,char* file2,char* filename)
{
      record recd[20];
      int i,k;
      k=0;
      fstream f1,f2;
      f1.open(file1,ios::in);
      f2.open(file2,ios::in);
      while(!f1.eof())
      {
            f1.getline(recd[k].name,20,'|');
            f1.getline(recd[k++].usn,20,'\n');
      }
      while(!f2.eof())
      {
            f2.getline(recd[k].name,20,'|');
            f2.getline(recd[k++].usn,20,'\n');
      }
      int t,y;
      record temp;
      for(t=0;t<k-2;t++)
      for(y=0;y<k-t-2;y++)
      if(strcmp(recd[y].name,recd[y+1].name)>0)
      {
            temp=recd[y];
            recd[y]=recd[y+1];
            recd[y+1]=temp;
      }
      fstream temp1;
```

```
	temp1.open(filename,ios::out);
	for(t=1;t<k-1;t++)
	temp1<<recd[t].name<<"|"<<recd[t].usn<<"\n";
	f1.close();
	f2.close();
	temp1.close();
	return;
}

void kwaymerge()
{
	int i,k;
	k=0;
	char filename[7][20]={"11.txt","22.txt","33.txt","44.txt","111.txt",
	                      "222.txt","1111.txt"};
	for(i=0;i<8;i+=2)
	{
		merge_file(fname[i],fname[i+1],filename[k++]);
	}
	k=4;
	for(i=0;i<4;i+=2)
	{
		merge_file(filename[i],filename[i+1],filename[k++]);
	}
	merge_file(filename[4],filename[5],filename[6]);
	return;
}


int main()
{
	int i;
	clrscr();
	cout<<"enter no of records\n";
	cin>>no;
	cout<<"\nenter the details\n";
	for(i=0;i<8;i++)
	file[i].open(fname[i],ios::out);
	for(i=0;i<no;i++)
	{
		cout<<"Name:";
		cin>>rec[i].name;
		cout<<"Usn:";
		cin>>rec[i].usn;
		file[i%8]<<rec[i].name<<"|"<<rec[i].usn<<"\n";
	}
	for(i=0;i<8;i++)
	file[i].close();
	kwaymerge();
```

```
	fstream result;
	result.open("1111.txt",ios::in);
	cout<<"sorted records\n";
	char name[20],usn[20];
	for(i=0;i<no;i++)
	{
		result.getline(name,20,'|');
		result.getline(usn,20,'\n');
		cout<<"\nName:"<<name<<"\nUsn:"<<usn<<"\n";
	}
	getch();
	return 0;
}
```

Output :

```
Enter the no. of records :4
Enter the details :

Name :rahul
USN  :25

Name :laxmi
USN  :16

Name :ajay
USN  :2

Name :deepak
USN  :8

Sorted Records :

Name :ajay
USN  :2

Name :deepak
USN  :8

Name :laxmi
USN  :16

Name :rahul
USN  :25
```

9. Write a C++ program to implement B-Tree for a given set of integers and its operations insert ( ) and search ( ). Display the tree.

B TREES : An Overview

- ✦ Multilevel indexes solving the problem of linear cost of insertion & deletion.
- ✦ Built upwards from the bottom rather than downward from the top.
- ✦ Order is the maximum number of immediate descendants that a node can have.
- ✦ Page is an index that holds many keys.
- ✦ Leaf is a page at the lowest level of the B Tree.
- ✦ Height is the longest simple path from the root to a leaf.
- ✦ Space Utilization is the ratio of amount of space used to hold the keys to the amount of space required to hold the B Tree.

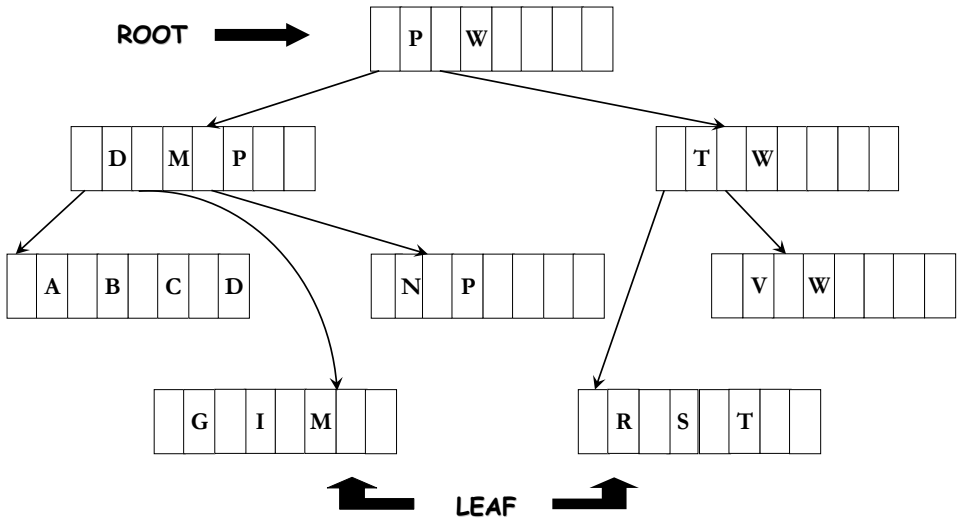
Properties

For a B Tree of Order *m*

- ◆ Every node has a maximum of *m* descendants
- ◆ Every node, except for root & leaves, has at least *m/2* descendants
- ◆ Root has at least 2 descendants

  (unless it is a leaf)
- ◆ All leaves are at the same level

B Tree of Order 4



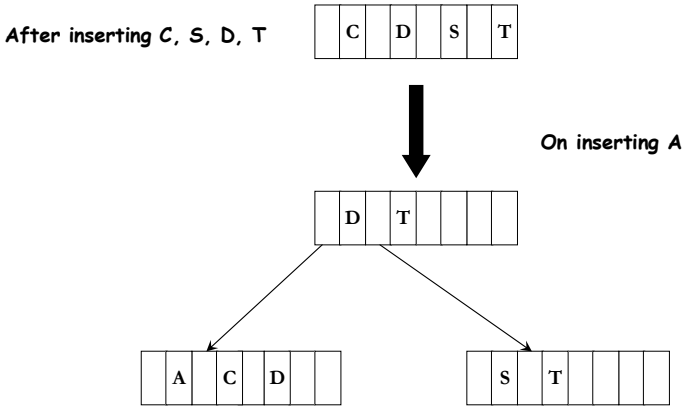
Operations on B Trees

- Insertion
- Deletion
- Searching

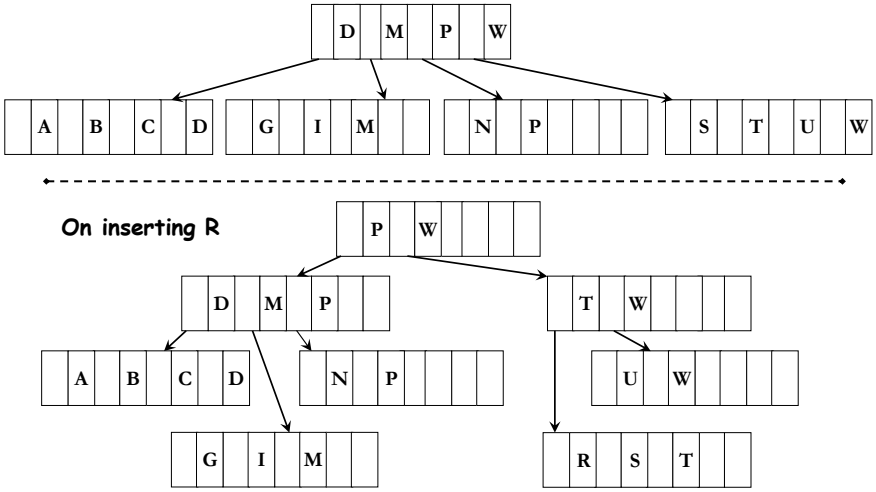
Insertion

- Begins with a search that proceeds all the way down to the leaf level.
- After finding the insertion location at the leaf level, the work of insertion, overflow detection & splitting proceeds upwards from the bottom.
- Create new root node, if current root was split.

No Split & Contained Split



Recursive Split



Searching

Recursive

Loads the page into memory & then searches through it.

Looks for key at successive lower levels of tree until it is found

Implementation

- *FindLeaf* : locates the leaf in which the key value is to be inserted or searched.
- *Split* : used to split the node into two when overflow occurs during insertion.

- *Insert* : adds new key value & displacement value of corresponding record into leaf level.
- *Search* : traverses the entire tree to locate if given key value is present.
- *Display* : displays B Tree structure.
- *Usage* : calculates the average space utilization of tree structure.
- *Driver* : creation of B Tree structure from existing student data file.

File_structure9.cpp

```
#include<iostream.h>
#include<stdio.h>
#include<fstream.h>
#include<stdlib.h>
#include<string.h>
class node
{
      public:
            int a[4];
            node *next[4];
            node *parent;
            int size;
            node();
};

node::node()
{
      for(int i=0;i<4;i++)
            next[i]=NULL;
            parent=NULL;
            size=0;
}

class btree
{
      node *root;
      public:
            node *findLeaf(int key,int &level);
            void updateKey(node *p,node *c,int newkey);
            void search(int key);
            void insert(int key);
            void insertIntoNode(node *n,int key,node *addresss);
            void promote(node *n,int key,node *addresss);
            node *split(node *n);
            void traverse(node *ptr);
            btree();
};

void btree::traverse(node *ptr)
{
      if(ptr==NULL)
      return;
      for(int i=0;i<ptr->size;i++)
            cout<<ptr->a[i]<<" ";
      cout<<endl;
      for(i=0;i<ptr->size;i++)
      traverse(ptr->next[i]);
}
```

```
btree::btree()
{
	root=NULL;
}

node* btree::findLeaf(int key,int &level)
{
	node *ptr=root;
	node *prevptr=NULL;
	level=0;
	int i;
	while(ptr)
	{
		i=0;
		level++;
		while(i<ptr->size-1 && key>ptr->a[i])
		i++;
		prevptr=ptr;
		ptr=ptr->next[i];
	}
	return prevptr;
}

node * btree::split(node *n)
{
	int midpoint=(n->size+1)/2;
	int newsize=n->size-midpoint;
	node *newptr=new node;
	node *child;
	newptr->parent=n->parent;
	int i;
	for(i=0;i<midpoint;i++)
	{
		newptr->a[i]=n->a[i];
		newptr->next[i]=n->next[i];
		n->a[i]=n->a[i+midpoint];
		n->next[i]=n->next[i+midpoint];
	}
	n->size=midpoint;
	newptr->size=newsize;
	for(i=0;i<n->size;i++)
	{
		child=n->next[i];
		if(child!=NULL)
			child->parent=n;
	}
	for(i=0;i<newptr->size;i++)
	{
```

```
            child=newptr->next[i];
            if(child!=NULL)
                  child->parent=newptr;
      }
      return newptr;
}

void btree::updateKey(node *parent,node *child,int newkey)
{
      if(parent==NULL)
            return;
      if(parent->size==0)
            return;
      int oldkey=child->a[child->size-2];
      for(int i=0;i<parent->size;i++)
            if(parent->a[i]==oldkey)
            {
                  parent->a[i]=newkey;
                  parent->next[i]=child;
            }
}

void btree::insertIntoNode(node *n,int key,node *address)
{
      int i;
      if(n==NULL)
            return;
      for(i=0;i<n->size;i++)
            if(n->a[i]==key)
                  return;
      i=n->size-1;
      while(i>=0 && n->a[i]>key)
      {
            n->a[i+1]=n->a[i];
            n->next[i+1]=n->next[i];
            i--;
      }
      i++;
      n->a[i]=key;
      n->next[i]=address;
      n->size++;
      if(i==n->size-1)
            updateKey(n->parent,n,key);
}

void btree::promote(node *n,int key,node *address)
{
      if(n==NULL)
            return;
```

```
	if(n->size<4)
	{
		insertIntoNode(n,key,address);
		return;
	}
	if(n==root)
	{
		root=new node;
		n->parent=root;
	}
	node *newptr=split(n);
	node *t;
	if(key<n->a[0])
		t=newptr;
	else
		t=n;
	insertIntoNode(t,key,address);
	promote(n->parent,n->a[n->size-1],n);
	promote(newptr->parent,newptr->a[newptr->size-1],newptr);
}

void btree::insert(int key)
{
	if(root==NULL)
	{
		root=new node;
		root->a[root->size]=key;
		root->size++;
		return;
	}
	int level;
	node *leaf=findLeaf(key,level);
	int i;
	for(i=0;i<leaf->size;i++)
		if(leaf->a[i]==key)
		{
			cout<<"The key to be inserted already exists"<<endl;
			return;
		}
	promote(leaf,key,NULL);
	cout<<"------------------\n";
	traverse(root);
	cout<<"------------------\n";
}

void btree::search(int key)
{
	if(root==NULL)
	{
```

```
            cout<<"The tree does not exist"<<endl;
            return;
      }
      int level;
      node *leaf=findLeaf(key,level);
      int flag=0;
      for(int i=0;i<leaf->size;i++)
            if(leaf->a[i]==key)
            {
                  flag=1;
                  cout<<"The  key  "<<key<<"  exists  in  the  B-tree  at  the  level
"<<level<<endl;
            }
      if(!flag)
            cout<<"The key searched for was not found"<<endl;
}

int main()
{
      btree b;
      int choice=1,key;
      while(choice<=2)
      {
            cout<<"1.Insert a key\n";
            cout<<"2.Search a key\n";
            cout<<"3.Exit\n";
            cout<<"Enter your choice: ";
            cin>>choice;
            switch(choice)
            {
                  case 1: cout<<"Enter the key to be inserted in a B-tree\n";
                        cin>>key;
                        b.insert(key);
                        break;
                  case 2: cout<<"Enter the key to be searched\n";
                        cin>>key;
                        b.search(key);
                        break;
            }
      }
      return 0;
}
```

Output :

```
1.Insert a Key
2.Search a key
3.Exit
Enter u'r choice :1

Enter The Key to be inserted in B-Tree
100

1.Insert a Key
2.Search a key
3.Exit
Enter u'r choice :1

Enter The Key to be inserted in B-Tree
50
-----------
50 100
-----------
1.Insert a Key
2.Search a key
3.Exit
Enter u'r choice :1

Enter The Key to be inserted in B-Tree
75
-------------
50 75 100
-------------
1.Insert a Key
2.Search a key
3.Exit
Enter u'r choice :1

Enter The Key to be inserted in B-Tree
200
---------------
50 75 100 200
---------------

1.Insert a Key
2.Search a key
3.Exit
Enter u'r choice :2

Enter The key to be searched
100
Key 100 exists in B-tree at level 1
```

10. Write a C++ program to implement B+ tree for a given set of integers and its operations insert ( ), and search ( ). Display the tree.

B+ tree structure is designed mainly in order to overcome the disadvantages of the B tree structure. The functionality of a B+ tree structure is although is similar to the behavior B tree structure, the flexibility and efficiency wise B+ tree structure is far better than its counterpart.

Basically B+ tree is a combination of B tree structure and the sequence structure. The B tree structural nodes are known as the index set and the sequence structure are known as the sequence set.

In all B-tree type structures, key search proceeds from the root downwards, following pointers to the nodes which contain the appropriate range of keys, as indicated by the reference values. Likewise, all B-trees grow from the leaves up. After obtaining the appropriate location for the new entry, it is inserted. If the node becomes overfull it splits in half and a pointer to the new half is returned for insertion in the parent node, which if full will in turn split, and so on.

B+trees distinguish **internal and leaf nodes**, keeping data only at the **leaves,** whereas ordinary B-trees would also **store keys in the interior**. B+tree insertion, therefore, requires managing the interior node reference values in addition to simply finding a spot for the data, as in the simpler B-tree algorithm.

B+tree algorithms incorporate an **insertion overflow** mechanism to enforce higher node utilization levels. B+tree insertion at full nodes may avoid splitting by first checking neighboring nodes. Keys from the full node are redistributed to a less full neighbor. If both neighbors are full, however, the split must take place.

Sequence set:

A sequence set is a **linked structure** of many number of sequence nodes. Each node of a sequence set contains **key-reference pairs**, which are sequenced based on key and a pointer, which points to its next successor. All the nodes are linked such that, when we start off at the first sequence node and if traverse through all the nodes, we can access all the records of the data file in ascending or descending order of key. The sequence set is considered as the **leaf level nodes** of the B+ tree structure.

Index set:

An index set structure of B+ tree is quite similar to the B tree structure. In B tree nodes, for every **key there is a corresponding reference value**. But in B+ tree index node, if the number of keys or separators is n, then number of reference values is always (n+1).

Here the separator **is the key value** which helps in distinguishing either index nodes or the sequence nodes and also the keys in combination with the reference values provide a path in order to search for a particular records present at the leaf level of B+ tree structure.

File_structure10.cpp

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

struct node
{
      int elements[50];
      node *link[5],*first,*next,*parent;
      int level;
};

int cur_level=0;

node *create_node()
{
      node *cur=new node;
      for(int i=0;i<5;i++)
      {
            cur->elements[i]=-1;
            cur->link[i]=NULL;
      }
      cur->next=NULL;
      cur->parent=NULL;
      cur->level=0;
      cur->first=NULL;
return cur;
}

node* search(node *root,int key,int depth)
{
      node *cur=root;
      if(depth==0)
            return root;
      int i=0,j;
      while(i<depth)
      {
            for(j=0;j<=4;j++)
            {
                  if(cur->elements[j]==-1)
                  {
                        cur=cur->link[j-1];
                        break;
                  }
                  if(key<cur->elements[j])
                  {
                        if(j==0)
                              cur=cur->first;
                        else
```

```
                              cur=cur->link[j-1];
                        break;
                  }
            }
            i++;
      }
      return cur;
}

int search(node *root,int key,int *index,int &k)
{
      node *cur=root;
      int i=0,j;
      while(i<cur_level)
      {
            for(j=0;j<4;j++)
            {
                  if(cur->elements[j]==-1)
                  {
                        if(j==0)
                              return 0;
                        cur=cur->link[j-1];
                        index[k++]=j;
                        break;
                  }
                  if(key<cur->elements[j])
                  {
                        if(j==0)
                        {
                              cur=cur->first;
                              index[k++]=0;
                        }
                        else
                        {
                              cur=cur->link[j-1];
                              index[k++]=j;
                        }
                        break;
                  }
            }
            i++;
      }
      for(j=0;j<4;j++)
            if(key==cur->elements[j])
            {
                  index[k++]=j;
                  return 1;
            }
      return 0;
```

```
}

void update_level(node *cur)
{
	cur->level++;
	if(cur->first!=NULL)
		update_level(cur->first);
	for(int i=0;i<5;i++)
		if(cur->link[i]!=NULL)
			update_level(cur->link[i]);
}

void display(node *root)
{
	node *q[100],*cur;
	int r=-1,f=0,i=0,j=0;
	q[++r]=root;
	cout<<"\nThe tree: \n";
	cout<<"\nLevel: "<<i<<endl;
	while(r>=f)
	{
		cur=q[f++];
		if(cur->level!=i)
			cout<<"\nLevel: "<<++i<<endl;
		if(cur->first!=NULL)
			q[++r]=cur->first;
		for(j=0;j<4;j++)
		{
			if(cur->elements[j]!=-1)
			{
				cout<<" "<<cur->elements[j];
				if(cur->link[j]!=NULL)
					q[++r]=cur->link[j];
			}
			else
				break;
		}
		cout<<"\t";
	}
}

node *insert(node *root,int key,int level,int type,node *child1,node *child2)
{
	node *cur=search(root,key,level);
	int i,j,flag=0;
	for(i=0;i<5;i++)
		if(cur->elements[i]==-1)
			break;
		if(i==4)
```

```
            flag=1;
      for(j=i;j>0;j--)
      {
            if(key<cur->elements[j-1])
            {
                  cur->elements[j]=cur->elements[j-1];
                  cur->link[j]=cur->link[j-1];
            }
            else
            {
                  cur->elements[j]=key;
                  cur->link[j]=child1;
                  if(child1!=NULL)
                        child1->parent=cur;
                  break;
            }
      }
      if(j==0)
      {
            cur->elements[j]=key;
            cur->link[j]=child1;
            cur->first=child2;
            if(child1!=NULL)
                  child1->parent=cur;
            if(child2!=NULL)
                  child2->parent=cur;
      }
      if(flag==1)
      {
            if(cur->level==0)
            {
                  cur_level++;
                  node *new_root=create_node();
                  new_root->first=cur;
                  cur->parent=new_root;
                  root=new_root;
                  update_level(cur);
            }
            node *new_cur=create_node();
            int next_key=cur->elements[2];
            if(type==0)
            {
                  new_cur->next=cur->next;
                  cur->next=new_cur;
            }
            for(j=0;j<3;j++)
            {
                  new_cur->elements[j]=cur->elements[j+2];
                  new_cur->link[j]=cur->link[j+2];
```

```
            }
            if(type==1)
            {
                  for(j=0;j<2;j++)
                  {
                        new_cur->elements[j]=cur->elements[j+3];
                        new_cur->link[j]=cur->link[j+3];
                  }
                  new_cur->elements[2]=-1;
                  new_cur->link[2]=NULL;
                  new_cur->first=cur->link[2];
            }
            for(j=2;j<5;j++)
            {
                  cur->elements[j]=-1;
                  cur->link[j]=NULL;
            }
            if(new_cur->link[0]!=NULL    &&    new_cur->link[1]!=NULL    &&    new_cur-
>link[2]!=NULL)
            {
                  new_cur->link[0]->parent=new_cur;
                  new_cur->link[1]->parent=new_cur;
                  new_cur->first->parent=new_cur;
                  if(type==0)
                        new_cur->link[2]->parent=new_cur;
            }
            new_cur->level=cur->level;
            root=insert(root,next_key,new_cur->level-1,1,new_cur,cur);
      }
      return root;
}

void sequential(node *root)
{
      node *cur=root;
      while(cur!=NULL)
      {
            for(int i=0;i<4;i++)
            {
                  if(cur->elements[i]==-1)
                        break;
                  cout<<"\t"<<cur->elements[i];
            }
            cur=cur->next;
      }
}

Void  main()
{
```

```
	int ch,key,flag=0,k;
	clrscr();
	node *root=create_node();
	node *cur=root;
	for(;;)
	{
		cout<<"1.Insert 2.Search 3.Display 4.Sequential access\n";
		cout<<"Enter the choice: ";
		cin>>ch;
		switch(ch)
		{
			case 1: cout<<"Enter key to be inserted: ";
				cin>>key;
				root=insert(root,key,cur_level,0,NULL,NULL);
				break;
			case 2: cout<<"Enter key: ";
				cin>>key;
				int index[100];
				k=0;
				flag=search(root,key,index,k);
				if(flag)
				{
					cout<<"Element found,Path is: ";
					for(int i=0;i<k-1;i++)
						cout<<" "<<index[i];
				}
				else
					cout<<"Element not found\n";
					break;
			case 3: display(root);
				break;
			case 4: sequential(cur);
				break;
			default:exit(0);
		}
	}
}
```

Output:

```
1.Insert a Key
2.Search a key
3.Traverse Leaf
4.Exit
enter u'r choice : 1

Enter The Key to be inserted in B-Tree
100

1.Insert a Key
```

```
2.Search a key
3.Traverse Leaf
4.Exit
enter u'r choice : 1

Enter The Key to be inserted in B-Tree
50
--------
50 100
--------

1.Insert a Key
2.Search a key
3.Traverse Leaf
4.Exit
enter u'r choice : 1

Enter The Key to be inserted in B-Tree
200

----------
50 100 200
----------
1.Insert a Key
2.Search a key
3.Traverse Leaf
4.Exit
enter u'r choice : 1

Enter The Key to be inserted in B-Tree
75
---------------
50 75 100 200
---------------

1.Insert a Key
2.Search a key
3.Traverse Leaf
4.Exit
enter u'r choice : 2

Enter The key to be searched
300

The Key Searched for was not found
```

11. Write a C++ program to store and retrieve student data from file using hashing. Use any collision resolution technique

WHAT IS HASHING?

A *hash* function is like a black box which produces an address every time you drop in a key.

It is a function h(K) that transforms a key K into an address.

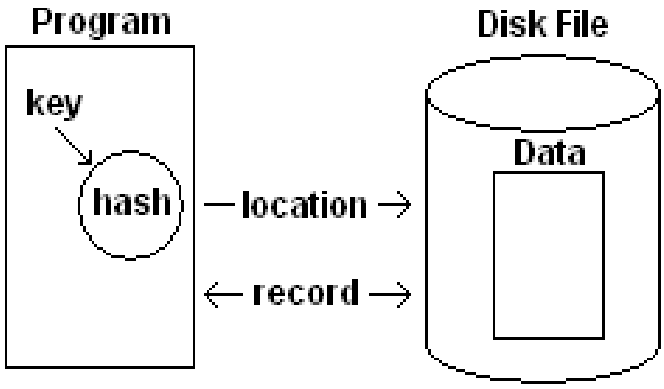
The addresses generated appear to be random hence hashing is also referred as *randomizing.*

The resulting address is used as the basis for storing and retrieving records.The key in our project is the student University Seat Number (USN)

Hashing

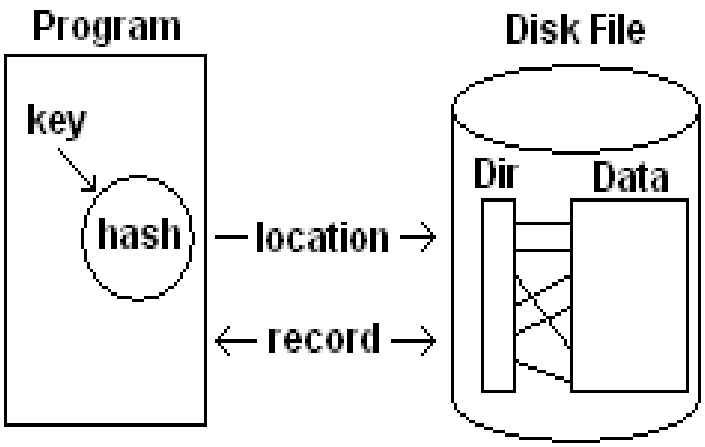
- How else can a single record in a large file be accessed?

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



Extendible Hashing

- How can hashing be applied go large, growing, files?
- **extendible hashing**
  An application of hashing that works well with files that over time undergo substantial changes in size.



HASHING ALGORITHM

REPRESENT THE KEY IN NUMERICAL FORM

If the key is a number, then this step is already accomplished.

For a string of characters, we take the ASCII code of each of the characters and use it to form a number.

For e.g.. LOWELL

76 79 87 69 76 76 32 32 32 32 32 32

L O W E L L | |

Blanks

In this algorithm we use the entire key rather than a part of it.

By using more parts of the key the probability of difference in address due to different keys is high.

FOLD AND ADD

76 79 | 87 69 | 76 76 | 32 32 | 32 32 | 32 32

- Integer values exceeding 32767 might cause an overflow
- Assuming the keys consist of only uppercase and blank characters, the largest addend is 9090
- These number pairs are considered as integer variables

- Hence the largest allowable intermediate result will be
  32767-9090=19937
- To ensure this ,we apply mod 19937 to the result

DIVIDE BY SIZE OF THE ADDRESS SPACE

In this step we divide the number produced in step-2 by the address size of the file.the remainder will be the home address of the file.

CREATING ADDRESSES

Because extendible hashing uses more bits of the hashed address as they are needed to distinguish between buckets, we need a function *makeaddress* that extracts a part of the full address.

It is also used to *reverse* the order of the bits in a hash address.

This is done because the LSBs of these integer values tend to have more *variation* than the MSBs

For e.g.. In a 4-bit address space we want to avoid the address of Bill ,Lee,Pauline to be 0000

Bill	0000	0011	0110	1100
Lee	0000	0100	0010	1000
Pauline	0000	1111	0110	0101

HASHING SCHEME

A function Hash ( ) generates the hash value for a key.

Make address ( ) reverses the order of bits to achieve more randomization.

File_structure11.cpp

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<iostream.h>
#include<fstream.h>
#include<new.h>
class node
{
 public:char name[20];
       char usn[20];
       node *link;
};
node *first=NULL;

void writeFile()
{
       node *p;
       char buffer[100];
       fstream out;
       out.open("student.txt",ios::out);
       if(!out)
       {
              cout<<"\nUnable to open the file student.txt in out mode";
              getch();
              exit(0);
       }
       p=first;
       while(p!=NULL)
       {
              strcpy(buffer,p->name);
              strcat(buffer,"|");
              strcat(buffer,p->usn);
              strcat(buffer,"\n");
              out<<buffer;
              p=p->link;
       }

}

void display()
{
       node *p;
       if(first==NULL)
       {
              cout<<"\nList is empty";
              return;
       }
```

```
	p=first;
	while(p!=NULL)
	{
		cout<<"|"<<p->name<<" "<<p->usn<<"|"<<"->";
		p=p->link;
	}

}

void Insert() //Insert the record at the rear end
{
	char name[20],usn[15];
	node *p,*q;
	cout<<"\nEnter name  = ";
	cin>>name;
	cout<<"\nEnter usn   = ";
	cin>>usn;
	p=new node;
	strcpy(p->name,name);
	strcpy(p->usn,usn);
	p->link=NULL;
	if(first==NULL)
	{
		first=p;
		writeFile();
		display();    //display the record on the screen
		return;
	}

	for(q=first;q->link!=NULL;q=q->link)
	{
	;
	}
	q->link=p;

	writeFile(); //writing the record to the file
	display();   //display the records to the screen.

}
void Delete()
{
	char usn[15];
	node *curr,*prev,*del;
	if(first==NULL)
	{
		printf("\nThe list is empty. Deletion is not possible");
		return;
	}
```

```
	cout<<"\nEnter the usn to be deleted = ";
	cin>>usn;
	if(strcmp(first->usn,usn)==0)
	{
		cout<<"\nRecord deleted";
		del=first;
		delete del;
		first=first->link;
		writeFile();
		return;
	}

	prev=NULL;
	curr=first;
	while( ( strcmp(curr->usn,usn) != 0 ) && curr!=NULL)
	{
		prev=curr;
		curr=curr->link;
	}
	if(curr==NULL)
	{
		cout<<"\nThe student with usn "<<usn<<" is not present";
		return;
	}

	prev->link=curr->link;
	writeFile();
	display();
}
void main()
{
	int choice;
	clrscr();
	for(;;)
	{
		printf("\n1:Insert_rear");
		printf("\n2:Delete_id");
		printf("\n3:exit");
		printf("\nEnter the choice=");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:Insert();
			       break;
			case 2:Delete();
			       break;
			case 3:exit(0);
			default:cout<<"\nInvalid option";
			break;
```

```
            }
      }
}
```

Output :

```
1:insert
2:retrive
3:exit

Enter the choice – 1

Enter the name – ajay
Enter the usn - 10

1:insert
2:retrive
3:exit

Enter the choice – 1

Enter the name – rahul
Enter the usn - 20


1:insert
2:retrive
3:exit

Enter the choice – 1

Enter the name – deepak
Enter the usn - 30


1:insert
2:retrive
3:exit

Enter the choice – 2

Enter the usn =  20

Record found : rahul 20
```

12. Write a C++ program to reclaim the free space resulting from the deletion of records using linked lists.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<iostream.h>
#include<fstream.h>
//#include<new.h>
class node
{
 public:char name[20];
       char usn[20];
       node *link;
};

node *first=NULL;

void writeFile()
{
       node *p;
       char buffer[100];
       fstream out;
       out.open("student.txt",ios::out);
       if(!out)
         {
             cout<<"\nUnable to open the file student.txt in out mode";
             getch();
             exit(0);
         }
       p=first;
       while(p!=NULL)
         {
           strcpy(buffer,p->name);
           strcat(buffer,"|");
           strcat(buffer,p->usn);
           strcat(buffer,"\n");
           out<<buffer;
           p=p->link;
         }
}
void display()
{
       char name[10],usn[10];
       fstream ifile;
       ifile.open("student.txt",ios::in);
       while(!ifile.eof())
       {
```

```
            ifile.getline(name,10,'|');
            ifile.getline(usn,10,'\n');
            cout<<name<<"\t"<<usn<<endl;
      }

 }
void Insert() //Insert the record at the rear end
{
      char name[20],usn[15];
       node *p,*q;
       cout<<"\nEnter name  = ";
       cin>>name;
       cout<<"\nEnter usn   = ";
       cin>>usn;
       p=new node;
       strcpy(p->name,name);
       strcpy(p->usn,usn);
       p->link=NULL;

       if(first==NULL)
       {
         first=p;
         writeFile();
         return;
       }
       for(q=first;q->link!=NULL;q=q->link);
       q->link=p;
       writeFile();

}
void Delete()
{
        char usn[15];
        node *curr,*prev,*del;
            if(first==NULL)
             {
              cout<<"Deletion is not possible"<<endl;
              return;
             }
        cout<<"\nEnter te usn to be deleted = ";
        cin>>usn;

            if(strcmp(first->usn,usn)==0)
             {
              cout<<"\nRecord deleted";
              del=first;
              delete del;
              first=first->link;
              writeFile();
```

```
                return;
               }

        prev=NULL;
        curr=first;
        while( ( strcmp(curr->usn,usn) != 0 ) && curr!=NULL)
               {
                 prev=curr;
                 curr=curr->link;
               }

        if(curr==NULL)
               {
                  cout<<"\nThe student with usn "<<usn<<" is not present";
                  return;
               }

        prev->link=curr->link;
        writeFile();
}
void main()
{
       int choice;
       clrscr();
       for(;;)
        {
               cout<<"\n1:Insert"<<endl;
               cout<<"\n2:Delete_id"<<endl;
               cout<<"\n3:display"<<endl;
               cout<<"\n4:exit"<<endl;
               cout<<"\nEnter the choice="<<endl;
               cin>>choice;
               switch(choice)
                     {
                      case 1:Insert();
                           break;
                      case 2:Delete();
                           break;
                      case 3:display();
                         break;
                      case 4:exit(0);
                     default:cout<<"\nInvalid option";
                         break;
                     }
        }
}
```

Output :

```
1-Insert_rear
```

```
2-Delete_id
3-Exit
Enter choice : 1

Enter name : ajay
Enter USN : 10

|ajay 10|->
1-Insert_rear
2-Delete_id
3-Exit
Enter choice : 1

Enter name : rahul
Enter USN : 20

|ajay 10|rahul 20|->

1-Insert_rear
2-Delete_id
3-Exit
Enter choice : 1

Enter name : deepak
Enter USN  : 30

|ajay 10|rahul 20|Deepak 30|->

1-Insert_rear
2-Delete_id
3-Exit
Enter choice :2


Enter the usn to be deleted = 20
|ajay 10|Deepak 30|->
```