File Handling in C++

Dr. Pradyumna Kumar Tripathy
Associate Professor, Dept. of Computer Sscience & Engineering,
Silicon Institute of Technology, Bhubaneswar

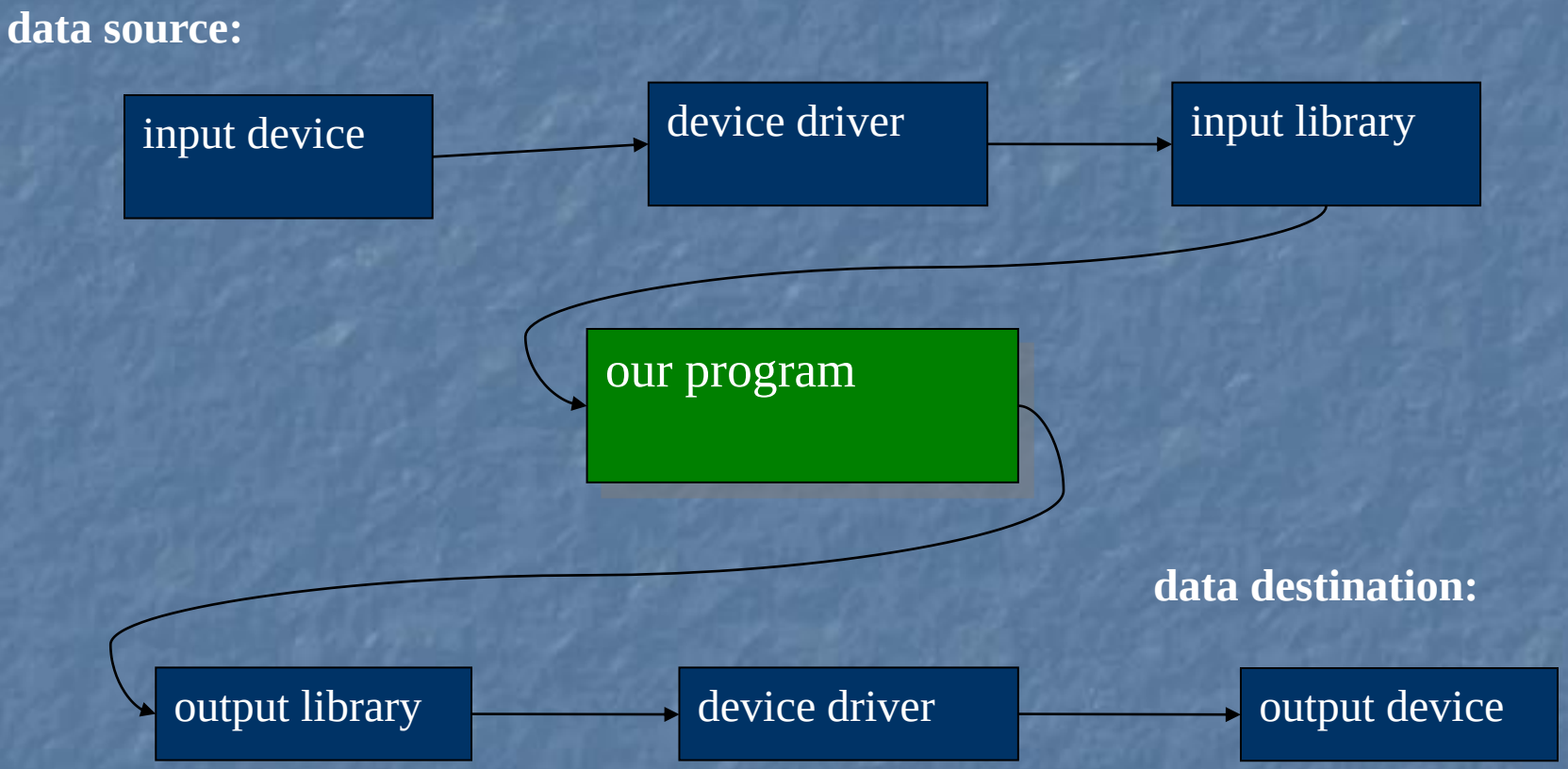
Abstract

- We get data from files, sensors, web connections, etc., which we want to analyze, print, graph, etc. Sometimes, we want to produce such data. In this lecture, we look at C++'s basic mechanisms for reading and writing streams of data.

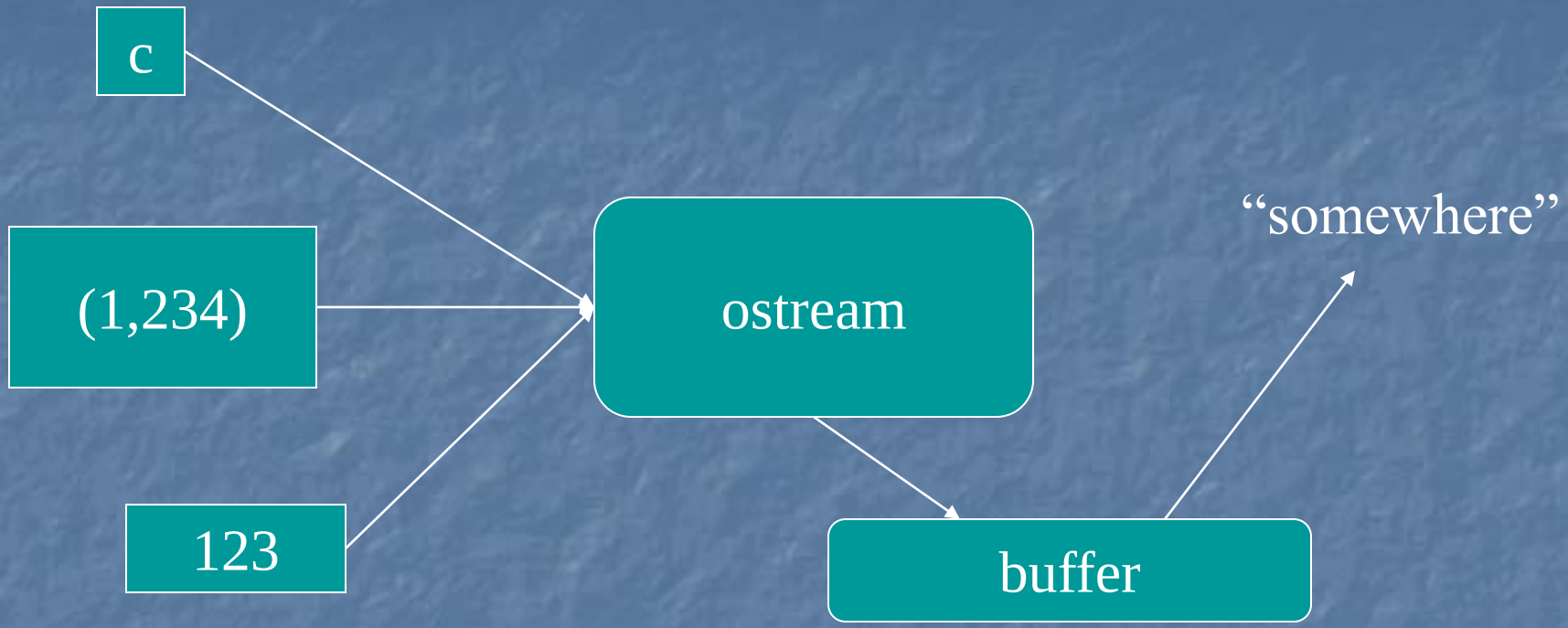
Overview

- Fundamental I/O concepts
- Files
  - Opening
  - Reading and writing streams
- I/O errors
- Binary File Handling
- Random Access

Input and Output

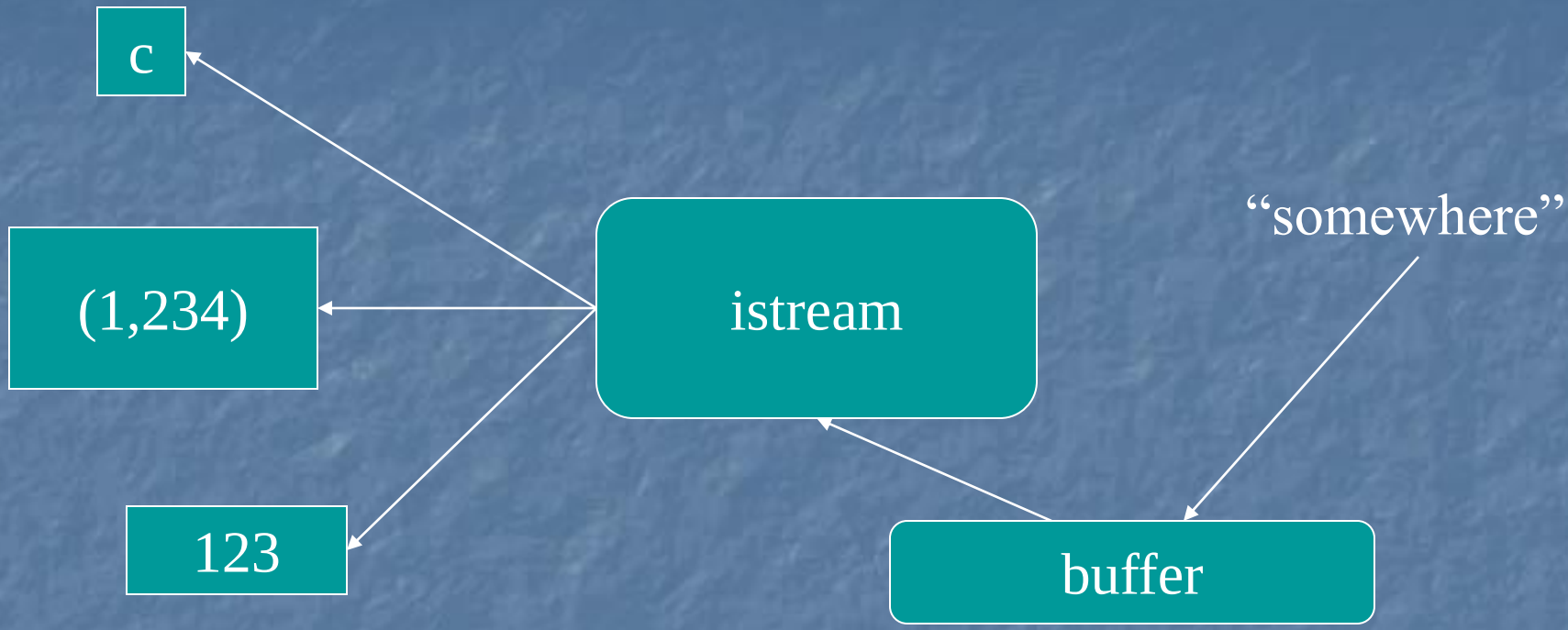


The stream model



- An **ostream**
  - turns values of various types into character sequences
  - sends those characters somewhere
    - *E.g.*, console, file, main memory, another computer

The stream model



- An **istream**
  - turns character sequences into values of various types
  - gets those characters from somewhere
    - *E.g.*, console, file, main memory, another computer

The stream model

- Reading and writing
  - Of typed entities
    - << (output) and >> (input) plus other operations
    - Type safe
    - Formatted
  - Typically stored (entered, printed, etc.) as text
    - But not necessarily (see binary streams in chapter 11)
  - Extensible
    - You can define your own I/O operations for your own types
  - A stream can be attached to any I/O or storage device

Files

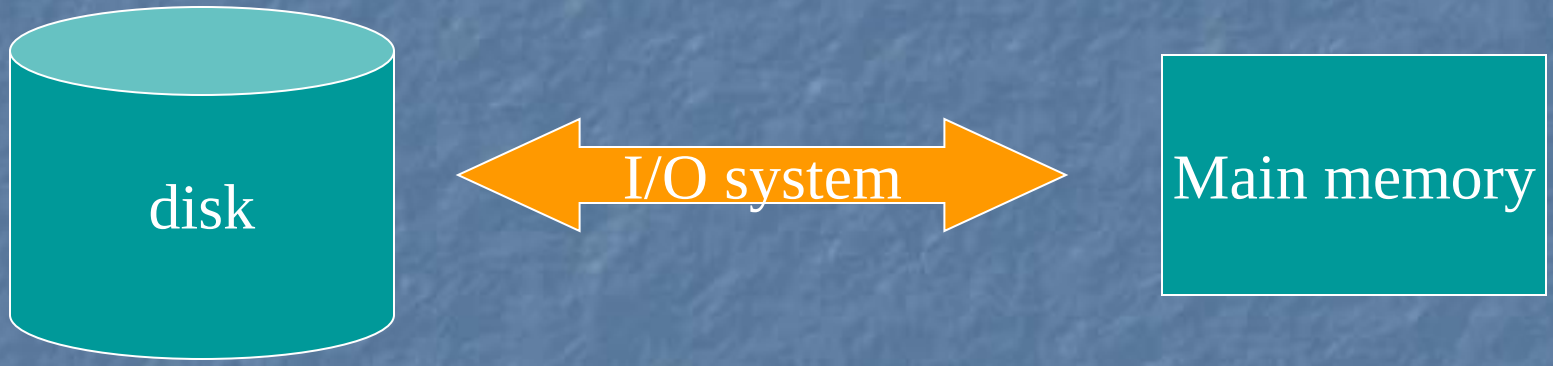
- We turn our computers on and off
  - The contents of our main memory is transient
- We like to keep our data
  - So we keep what we want to preserve on disks and similar permanent storage
- A file is a sequence of bytes stored in permanent storage
  - A file has a name
  - The data on a file has a format
- We can read/write a file if we know its name and format

A file

- At the fundamental level, a file is a sequence of bytes numbered from 0 upwards
- Other notions can be supplied by programs that interpret a “file format”
  - For example, the 6 bytes "123.45" might be interpreted as the floating-point number 123.45

Files

- General model



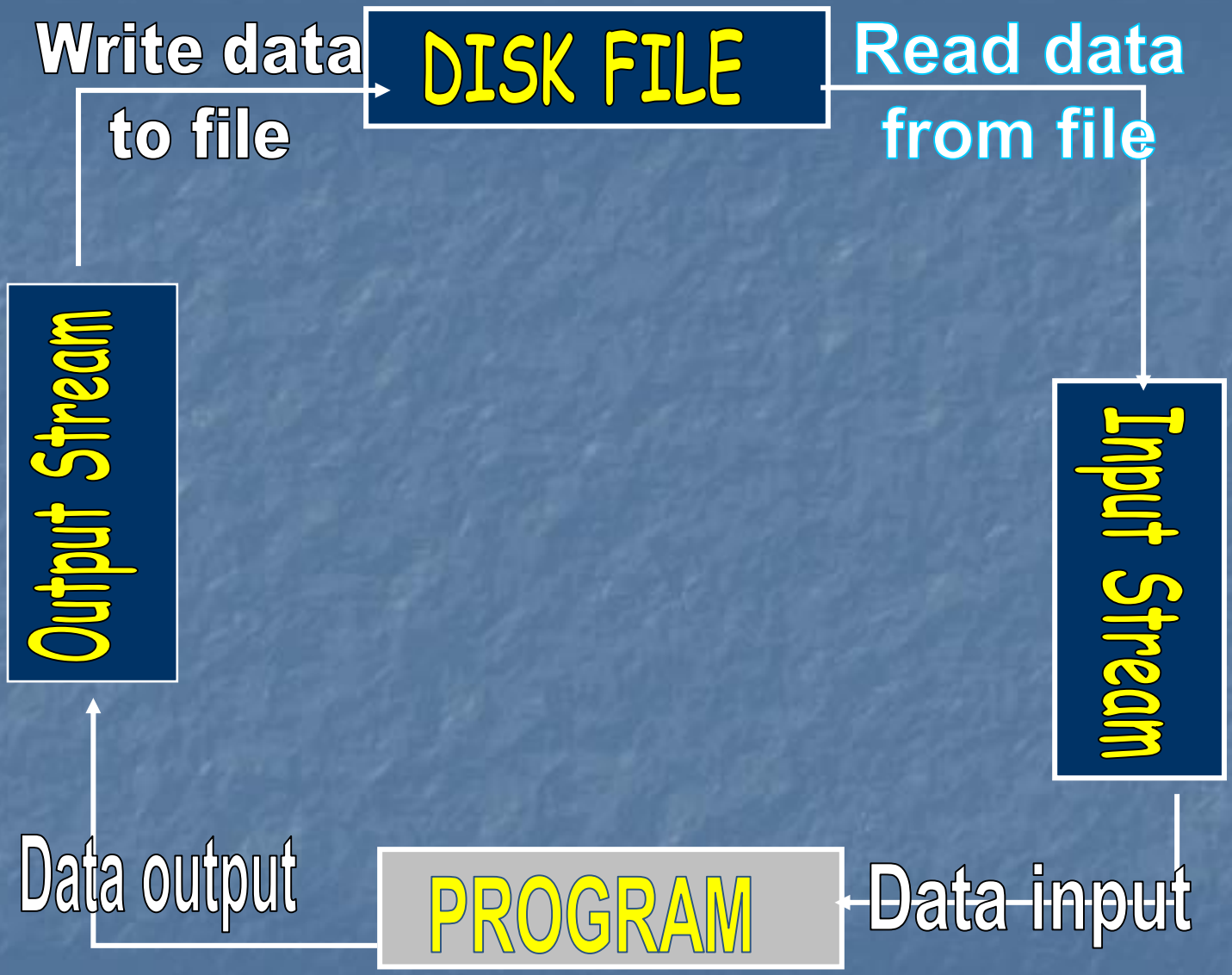
Files
(sequences of bytes)

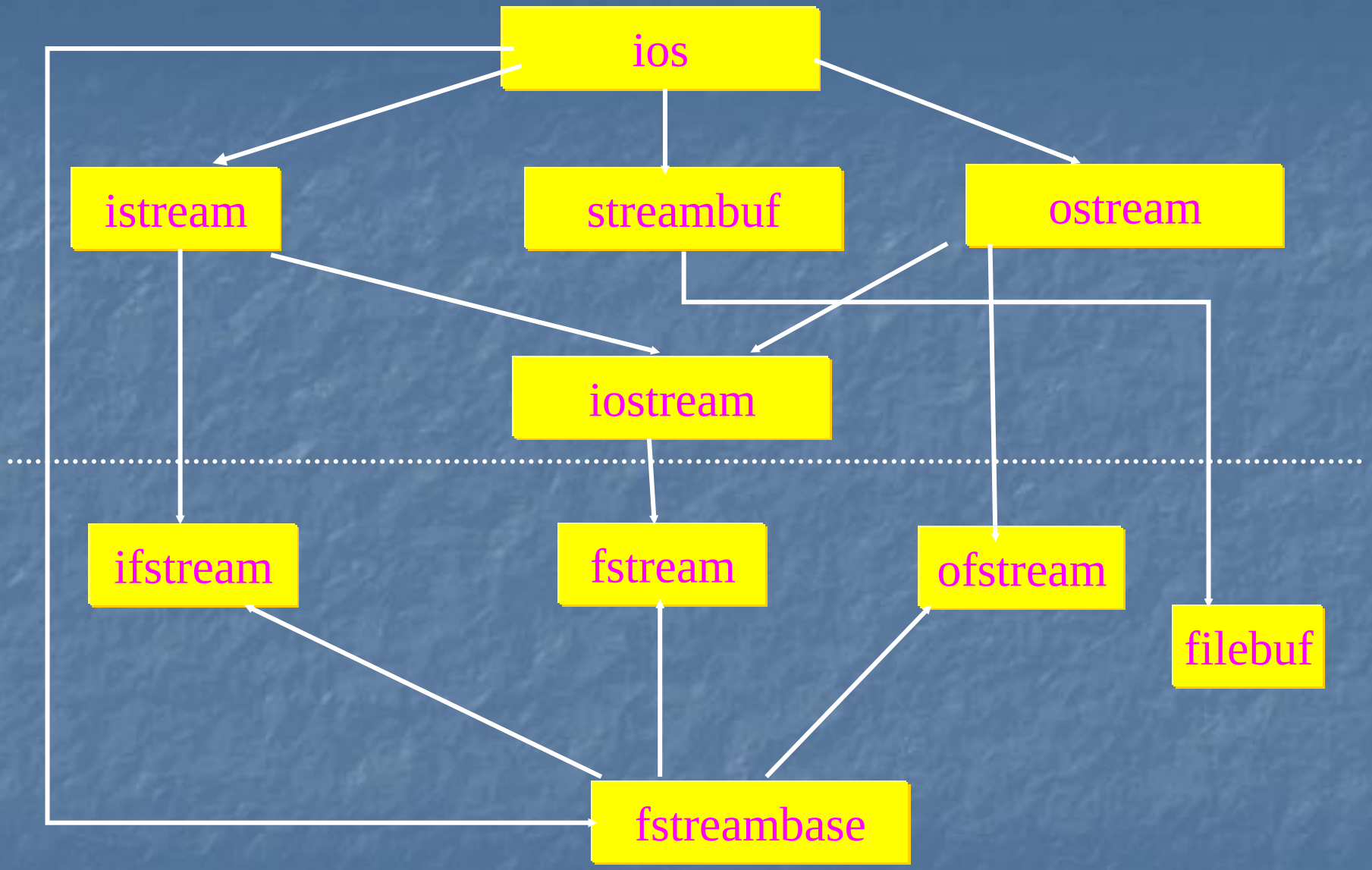
iostreams

Objects
(of various types)

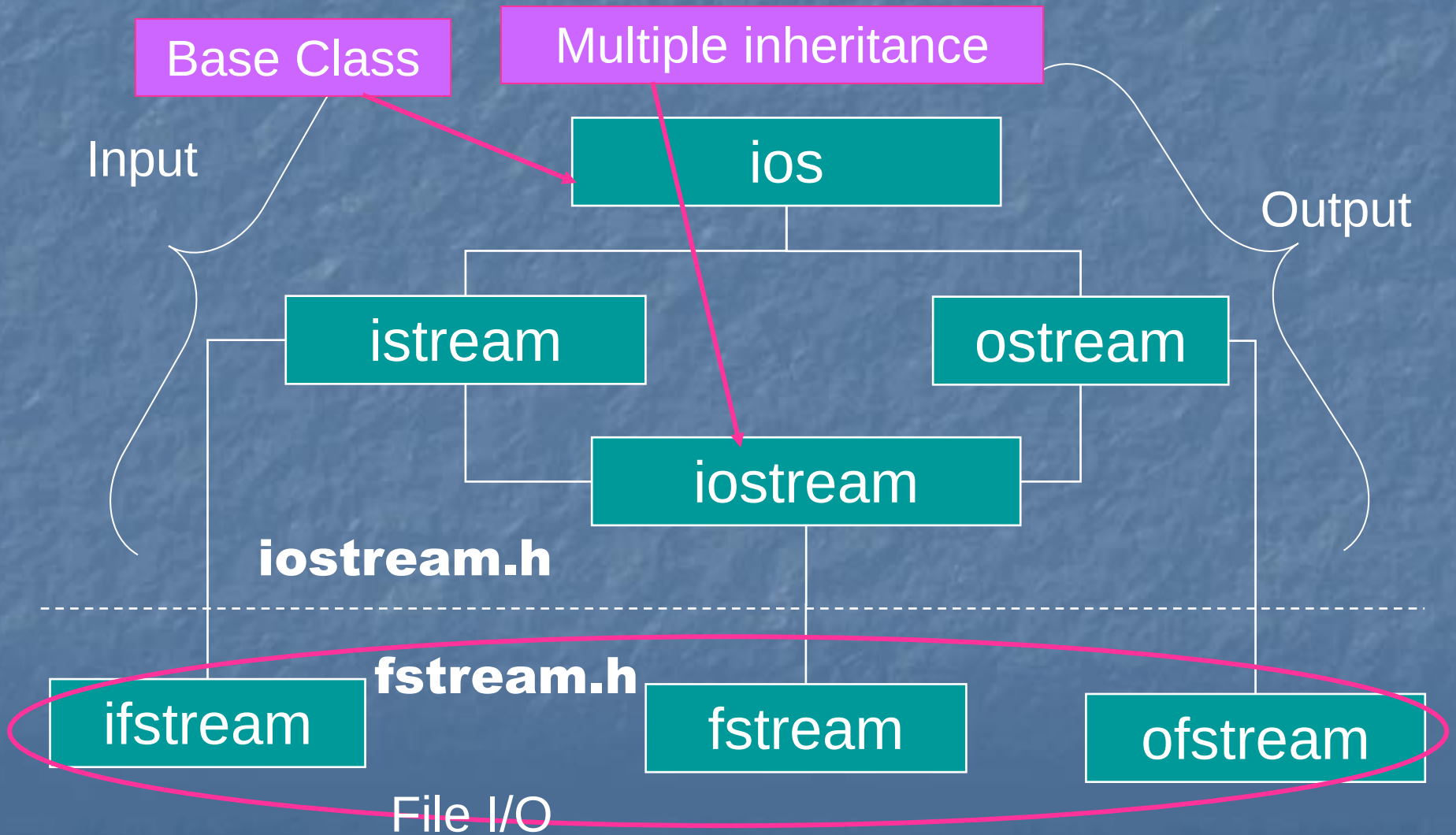
Files

- To read a file
  - We must know its name
  - We must open it (for reading)
  - Then we can read
  - Then we must close it
    - That is typically done implicitly
- To write a file
  - We must name it
  - We must open it (for writing)
    - Or create a new file of that name
  - Then we can write it
  - We must close it
    - That is typically done implicitly


Write data to file
DISK FILE
Read data from file
Output Stream
Input Stream
Data output
PROGRAM
Data input


ios
istream
streambuf
ostream
iostream
ifstream
fstream
ofstream
filebuf
fstreambase

File Class Hierarchy



Stream I/O Classes & Objects

- >> and << are right and left bit shift operators
- Have been overloaded for input and output
- **`cin`** is an object of class istream
  - tied to standard input, keyboard
- **`cout`** is an object of class ostream
  - tied to standard output, screen

C++ Files and Streams

- C++ views each files as a sequence of bytes.
- Each file ends with an ***end-of-file*** marker.
- When a file is ***opened***, an object is created and a stream is associated with the object.
- To perform file processing in C++, the header files **<iostream.h>** and **<fstream.h>** must be included.
- <fstream.> includes <ifstream> and <ofstream>

FUNCTIONS OF FILE STREAM CLASSES

- **filebuf** – It sets the buffer to read and write, it contains close() and open() member functions on it.
- **fstreambase** – this is the base class for fstream and, ifstream and ofstream classes. therefore it provides the common function to these classes. It also contains open() and close() functions.
- **ifstream** – Being input class it provides input operations it inherits the functions get( ), getline( ), read( ), and random access functions seekg( ) and tellg( ) functions.
- **ofstream –** Being output class it provides output operations it inherits put( ), write( ) and random access functions seekp( ) and tellp( ) functions.
- **fstream** – it is an i/o class stream, it provides simultaneous input and output operations.

File TYPES

- A File can be stored in two ways
- **Text File**
- **Binary File**

Text Files : Stores information in ASCII characters. In text file each line of text is terminated by with special character known as EOL (End of Line) In text file some translations takes place when this EOL character is read or written.

Binary File: it contains the information in the same format as it is held in the memory. In binary file there is no delimiter for a line. Also no translation occur in binary file. As a result binary files are faster and easier for program to read and write.

File modes

The File Mode describes how a file is to be used ; to read from it, write to it, to append and so on

Syntax :

Stream_object.open(“filename”,mode);

File Modes :

- **ios::out:** It open file in output mode (i.e write mode) and place the file pointer in beginning, if file already exist it will overwrite the file.
- **ios::in** It open file in input mode(read mode) and permit reading from the file.

File modes

- **ios::app** It open the file in write mode, and place file pointer at the end of file i.e to add new contents and retains previous contents. If file does not exist it will create a new file.
- ios::ate It open the file in write or read mode, and place file pointer at the end of file i.e input/ output operations can performed anywhere in the file.
- ios::trunc It truncates the existing file (empties the file).
- ios::nocreate If file does not exist this file mode ensures that no file is created and open() fails.
- ios::noreplace If file does not exist, a new file gets created but if the file already exists, the open() fails.
- ios::binary Opens a file in binary mode.

Closing a File

A File is closed by disconnecting it with the stream it is associated with. The close() function is used to accomplish this task.

Syntax:

```
Stream_object.close( );
```

Example :

```
fout.close();
```

Steps To Create A File

1. Declare an object of the desired file stream class(ifstream, ofstream, or fstream)

2. Open the required file to be processed using constructor or open function.

3. Process the file.

4. Close the file stream using the object of file stream.

eof ( ) Function

This function determines the end-of-file by returning true(non-zero) for end of file otherwise returning false(zero).

Syntax

Stream_object.eof( );

Example :

fout.eof( );

File I/O Example: Open the file with validation

First Method (use the constructor)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
//declare and automatically open
the file
ofstream outFile(“fout.txt");
// Open validation
if(! outFile)
{
 cout << “Cannot open file.\n ”;
 return 1;
}
return 0;
}
```

Second Method (use Open function)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
//declare output file variable
ofstream outFile;
// open an exist file fout.txt
outFile.open(“fout.txt”);
// Open validation
if(! outFile.is_open() )
{
    cout << “Cannot open file.\n ”;
    return 1;
}
return 0;
}
```

Text File Functions

get() – read a single character from text file and store in a buffer.

e.g file.get(ch);

put() - writing a single character in textfile

e.g. file.put(ch);

getline() - read a line of text from text file store in a buffer.

e.g file.getline(s,80);

We can also use **file>>ch for reading and file<<ch writing in text file. But >> operator** does not accept white spaces.

File I/O Example: Reading

Read char by char

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{//Declare and open a text file
ifstream openFile(“data.txt");
 char ch;
 //do until the end of file
while( ! openFile.eof() )
{
OpenFile.get(ch); // get one character
cout << ch;   // display the character
}
openFile.close(); // close the file

    return 0;
}
```

Read a line

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{//Declare and open a text file
ifstream openFile("data.txt");
string line;
while(!openFile.eof())
{//fetch line from data.txt and put it in a string
getline(openFile, line);
cout << line;
}
openFile.close(); // close the file
return 0; }
```

File I/O Example: Writing

First Method (use the constructor)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{/* declare and automatically
open the file*/
ofstream outFile("fout.txt");

//behave just like cout, put
the word into  the file
outFile << "Hello World!";

outFile.close();

return 0;
}
```

Second Method (use Open function)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{// declare output file variable
ofstream outFile;
// open an exist file fout.txt
outFile.open("fout.txt”);

//behave just like cout, put the word
into  the file
outFile << "Hello World!";

outFile.close();

return 0;
}
```

Program to create a text file using strings I/O

```
#include <iostream>
#include <fstream>                          //header file for file operations
using namespace std;
int main()
{
char s[80], ch;
ofstream file("myfile.txt");                //open file in default output mode
do
{
        cout<<"\n enter line of text";
        gets(s);                            //standard input
        file<<s;                            // write in a file myfile.txt
        cout<<"\n more input y/n";
        cin>>ch;
} while(ch!='n'||ch!='N');

file.close();
}                                           //end of main
```

Program to read content of 'myfile.txt' and display it on monitor.

```
#include <iostream>
#include<fstream>                    //header file for file operations
using namespace std;
int main()
{
char ch;
ifstream file("myfile.txt");         //open myfile.txt in default input mode
while(file)
{
        file.get(ch);                // read a character from text file 'myfile.txt'
        cout<<ch;                    // write a character in text file 'myfile.txt '
}

file.close();
}                                    //end of main
```

Binary File Functions

read( )- read a block of binary data or reads a fixed number of bytes from the specified stream and store in a buffer.

Syntax : Stream_object.read((char *)& Object, sizeof(Object));

e.g file.read((char *)&s, sizeof(s));

write( ) – write a block of binary data or writes fixed number of bytes from a specific memory location to the specified stream.

Syntax : Stream_object.write((char *)& Object, sizeof(Object));

e.g file.write((char *)&s, sizeof(s));

Binary File Functions

Note: Both functions take two arguments.

- The first is the address of variable, and the second is the length of that variable in bytes. The address of variable must be type cast to type char*(pointer to character type)

- The data written to a file using write( ) can only be read accurately using read( ).

Program to create a binary file ‘student.dat’ using structure.

```
#include <iostream>
#include<fstream>
using namespace std;
struct student
{
        char name[15];
        float percent;
};
int main() {
ofstream fout;
char ch;
fout.open(“student.dat”, ios::out | ios:: binary);
student s;
if(!fout) {
        cout<<“File can’t be opened”;
        exit(0);
        }
```

```
do
{
cout<<”\n enter name of student”;
gets(s.name);
cout<<”\n enter percentage”;
cin>>s.percent;
fout.write((char *)&s, sizeof(s));  // writing record in student.dat file
cout<<”\n more record y/n”;
cin>>ch;
}while(ch!=’n’ || ch!=’N’);

fout.close();
return 0;
}
```

Program to read a binary file 'student.dat' display records on monitor.

```
#include <iostream>
#include <fstream>
struct student
{
        char name[15];
        float percent;
};
int main()
{
ifstream fin;
student s;
fin.open("student.dat",ios::in | ios:: binary);
fin.read((char *)&s, sizeof(student)); //read record from file
```

CONTD....

```
while(fin)
{
cout<<s.name;
cout<<“\n has the percent: ”<<s.percent;
fin.read((char *) &s, sizeof(student));
}
fin.close();
return 0;
}
```

File Pointer

The file pointer indicates the position in the file at which the next input/output is to occur.

Moving the file pointer in a file for various operations viz modification, deletion , searching etc. Following functions are used

seekg(): It places the file pointer to the specified position in input mode of file.

e.g

file.seekg(p,ios::beg); or

file.seekg(-p,ios::end), or

file.seekg(p,ios::cur)

i.e to move to p byte position from beginning, end or current position.

File Pointer

seekp(): It places the file pointer to the specified position in output mode of file.

e.g

```
file.seekp(p,ios::beg);
file.seekp(-p,ios::end);
file.seekp(p,ios::cur);
```

i.e to move to p byte position from beginning, end or current position.

tellg(): This function returns the current working position of the file pointer in the input mode.

e.g int p=file.tellg();

tellp(): This function returns the current working position of the file pointer in the output mode.

e.f int p=file.tellp();

I/O error handling

- Sources of errors
  - Human mistakes
  - Files that fail to meet specifications
  - Specifications that fail to match reality
  - Programmer errors
  - Etc.

- iostream reduces all errors to one of four states
  - **good()** *// the operation succeeded*
  - **eof()** *// we hit the end of input (“end of file”)*
  - **fail()** *// something unexpected happened*
  - **bad()** *// something unexpected and serious happened*

Sample integer read “failure”

- Ended by “terminator character”
  - 1 2 3 4 5 *
  - State is **fail()**
- Ended by format error
  - 1 2 3 4 5.6
  - State is **fail()**
- Ended by “end of file”
  - 1 2 3 4 5 end of file
  - 1 2 3 4 5 Control-Z (Windows)
  - 1 2 3 4 5 Control-D (Unix)
  - State is **eof()**
- Something really bad
  - Disk format error
  - State is **bad()**

```
#include <fstream>
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
class student {
int rollno;    char name[20];
char branch[3];   float marks;
char grade;
public:
void getdata()   {
cout<<"Rollno: "; cin>>rollno;
cout<<"Name: ";  cin>>name;
cout<<"Branch: "; cin>>branch;
cout<<"Marks: "; cin>>marks;
if(marks>=75) {grade = 'A';     }
else if(marks>=60){grade = 'B';}
else if(marks>=50){grade = 'C';}
else if(marks>=40){grade = 'D';}
else {grade = 'F'; }
}
```

```
void putdata()
{
cout<<"Rollno: <<rollno<<"\tName:
"<<name<<"\n";
cout<<"Marks:"<<marks<<"\tGrade
: "<<grade<<"\n";
}

int getrno()
{
        return rollno;
}

void modify();
}stud1, stud;
```

```
void student::modify() {
cout<<"Rollno: "<<rollno<<"\n";
cout<<"Name:"<<name<<"\tBranch:
"<<branch<<"\tMarks: "<<marks<<"\n";

cout<<"Enter new details.\n";
char nam[20]=" ", br[3]=" ";
float mks;
cout<<"New name:(Enter '.' to retain old
one): ";
cin>>name;
cout<<"New branch:(Press '.' to retain old
one): ";
cin>>br;
cout<<"New marks:(Press -1 to retain old
one): ";
cin>>mks;
```

```
if(strcmp(nam, ".")!=0){
strcpy(name, nam); }
if(strcmp(br, ".")!=0) {
strcpy(branch, br);   }
if(mks != -1)     {
marks = mks;
if(marks>=75){
grade = 'A';       }
else if(marks>=60) {
grade = 'B';       }
else if(marks>=50){
grade = 'C';       }
else if(marks>=40){
grade = 'D';       }
else{grade = 'F';}
}
}
```

```
int main()
{
fstream fio;
fio.open("mark.dat", ios::in | ios::out
| ios::binary);
if (!fio)
        cout<<"\n Cannot open file";
char ans='y';
fio.seekp(0);
while(ans=='y' || ans=='Y')
{
stud1.getdata();
fio.write((char *)&stud1,
sizeof(stud1));
cout<<"Record added to the file\n";
cout<<"\nWant to enter more ?
(y/n)..";
cin>>ans;
}
int rno;
long pos;
char found='f';
```

```
cout<<"Enter rollno of student
whose record is to be modified: ";
cin>>rno;
fio.seekg(0);
while(!fio.eof()){
fio.read((char *)&stud1,
sizeof(stud1));
cout<<"\n Read a record";
cin.get();
if(stud1.getrno() == rno)
{
  stud1.modify();
  fio.seekg(pos);
fio.write((char *)&stud1,
sizeof(stud1));
found = 't';
break;  }
}
```

```
if(found=='f')
{
cout<<"\nRecord not found in the file..!!\n";
cout<<"Press any key to exit...\n";
cin.get();
exit(2);
}

 fio.seekg(0);
 cout<<"Now the file contains:\n";
 while(!fio.eof())
       {
 fio.read((char *)&stud, sizeof(stud));
stud.putdata();
       }
fio.close();
cin.get();
return 0;
}
```

Thank You

Email: pradyumnatripathy @ gmail.com
ptripathy @silicon.ac.in