Fundamentals of Computer Programming

Chapter 8
File Handling (Management)

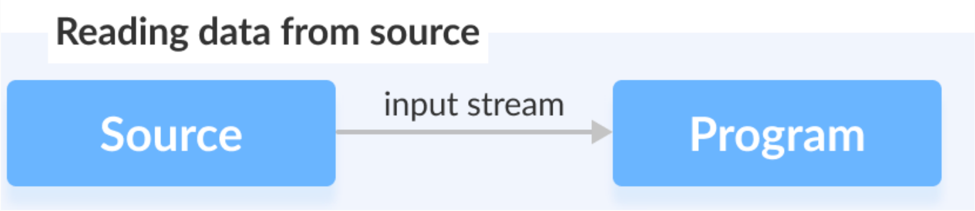
Chere L. (M.Tech)
Lecturer, SWEG, AASTU

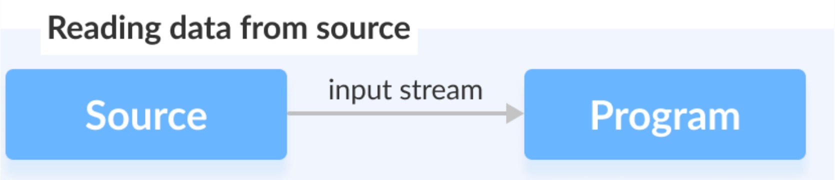
Outline

- Revision on I/O Stream
- Basics of File Management
    - *Types of file (Text file and Binary file)*
    - *File Stream*
- File Manipulation Process
    - *Opening and closing files*
    - *File Reading and Writing Operations*
    - *File Modes and Offset*
- File stream state functions
- Random File Access
- Practical Examples
- Practical Exercises

Revision on I/O Stream (1/3)

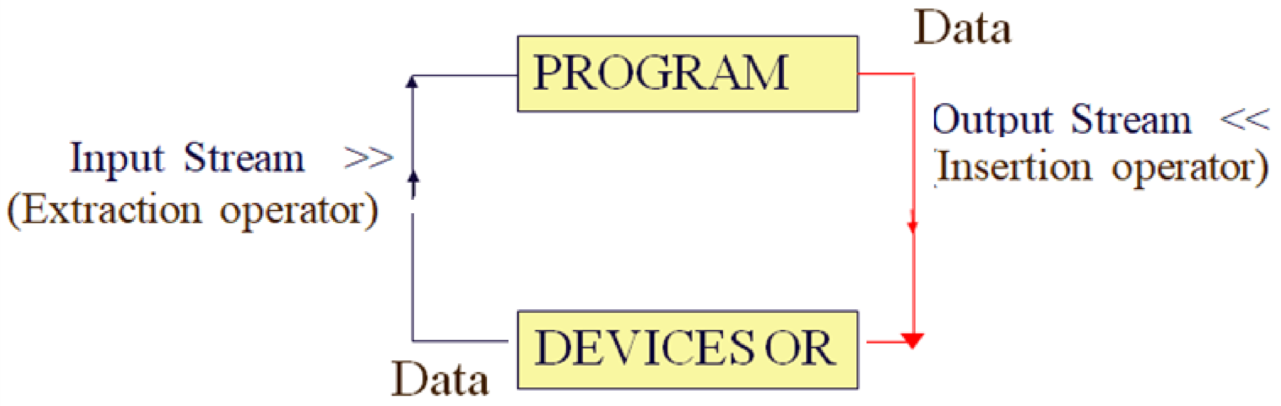
- The basic requirement of a program is/are ***I/O data/information***.
  - *i.e. Any program requires an* ***Input data*** *to be processed and should produce some kind of* ***Output data/information***
- The *flow of data* between your program and the source of data is referred as **Stream.**
- **Different streams** are used to *represent different kinds of data flow.*
- Mainly the stream is either *INPUT Stream* or *OUTPUT stream*.





Revision on I/O Stream (2/3)

- The primary source of program I/O data are **INPUT devices** and OUTPUT devices, mainly *keyboard* and *computer screen (monitor).*
- The flow of data between *I/O devices and a program* is called **Standard I/O Stream.**
- The two main standard I/O streams are
  - ***cin -*** *Input from stream object connected to keyboard*
  - ***cout -*** *Output to stream object connected to screen*



Revision on I/O Stream (3/3)

- **What is the draw backs of standard I/O streams?**
  - The data is Input from the keyboard where as the output data is print to the screen where an Ordinary variable is used.
  - The ordinary variables (even records and arrays) are kept in main memory which is **temporary** storage.
  - Moreover, the memory is limited in size (storage capacity).
  - As a result, the Input/output data would be lost as soon as you exit from the program.
- **Solution**
  - Store data on the secondary storage **permanently.**
  - Input/output data can be stored on secondary storage as a **file (records).**

Basics of File Management (1/8)

What is file?

- File is a collection of data or a set of characters or maybe a text or a program.
- File is stored permanently on secondary storage (disk) and can be retrieved when needed.

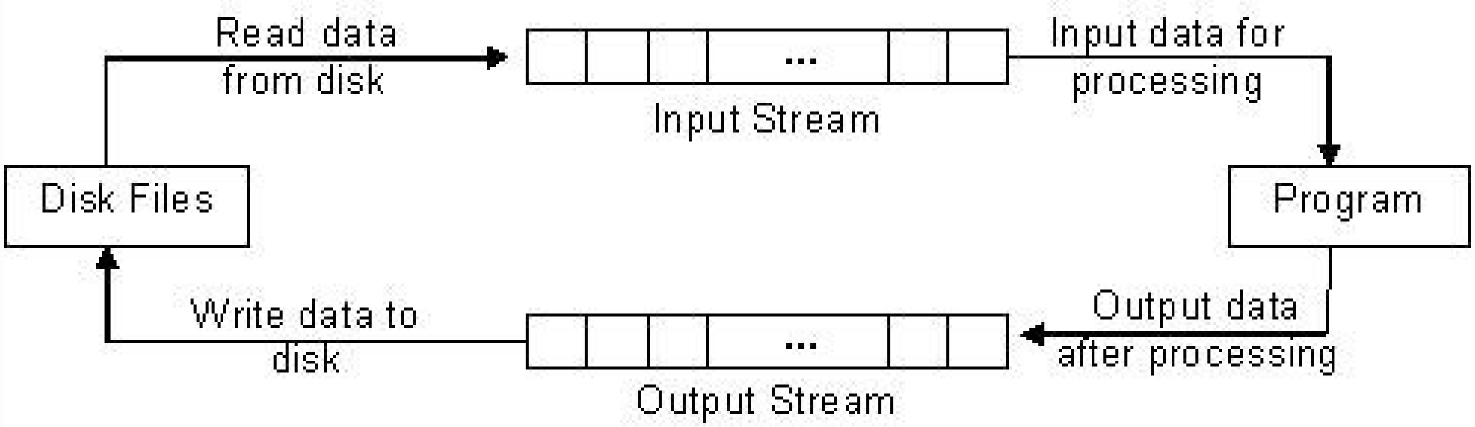
Why File?

- ✓ Used to store data relatively in ***permanent form***, on hard disk or other form of secondary storage.
- ✓ Files can ***hold huge amounts*** of data if it need be.

Basics of File Management (2/8)

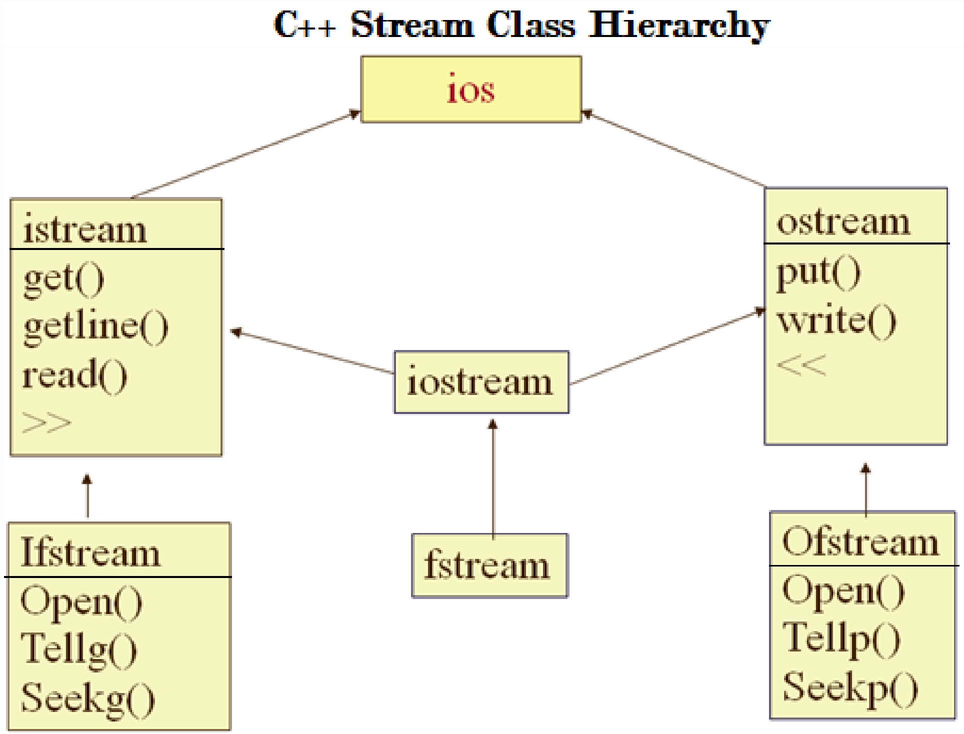
File Handling (Management)

- File handling is a *mechanism to store the output of a program* in a file and perform various operations on it.
- The flow of data between file and program is referred as **File stream**.
    - ***File Input Stream*** *– reads data from disk file to the program*
    - ***File output Stream*** *– writes data to the disk file from the program.*



Basics of File Management (3/8)

- Each file stream is associated with a *particular class*, which contains member functions and definitions for dealing with a particular kind of data flow.
- The I/O system of C++ provides the following *stream classes* which have access to standard I/O stream objects and file functions.

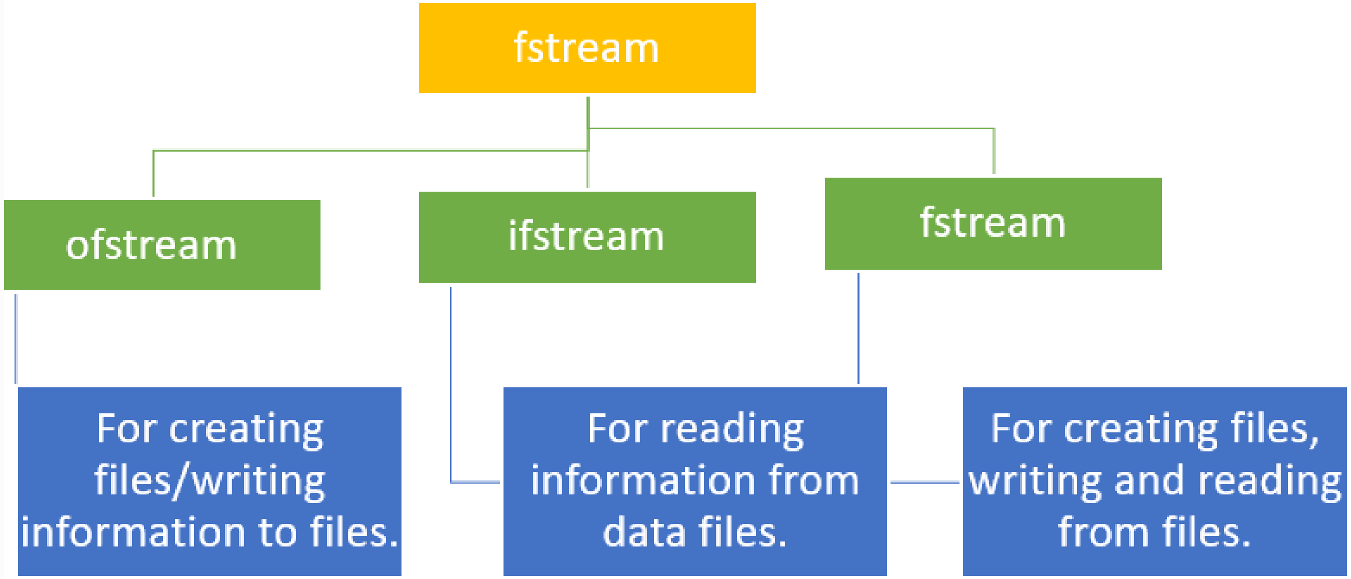


Note: *Upward arrows indicate the base calls and inheritance hierarchy*

Basics of File Management (4/8)

- The main class stream that used for file manipulation are
  - *ifstream – provides input operations on files*
  - *ofstream – provides output operations on files*
  - *fstream – supports both input and output operations on files.*

    - allow simultaneously to read from and write to a file



fstream

ofstream

ifstream

fstream

For creating files/writing information to files.

For reading information from data files.

For creating files, writing and reading from files.

Basics of File Management (5/8)

Types of File

- There are two types of file which the streams are linked to.

(a) Text File

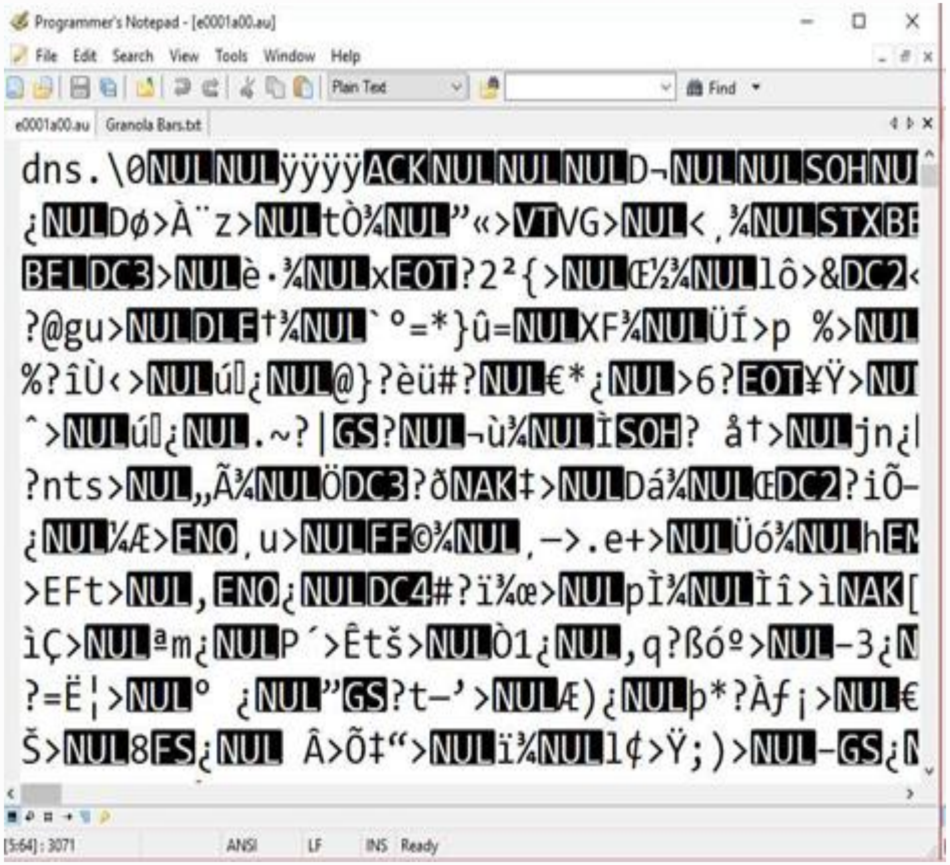
- *A sequence of readable characters separated by space and newline characters.*
- *Character translations may occur as required by the host environment*
- *There may not be a one-to-one relationship between the characters that are written (or read) and those on the external device.*

Basics of File Management (6/8)

(a) Binary File

- *A sequence of bytes with a one-to-one correspondence to those in the external device*
  - *The number of bytes written (or read) is the same as the number on the external device*
- No character translations required
  - *Data stored to disk in the same form in which it is represented in main memory.*
  - *As a result saving data in binary format is faster and takes less space*
- Human unreadable
  - *If you ever try to edit a binary file for example containing numbers you will see that the numbers appear as nonsense characters.*
  - *Do not normally use anything to separate the data into lines.*
- Binary format data can not easily transferred from one computer to another due to variations in the internal representation of the data fro computer to computer.

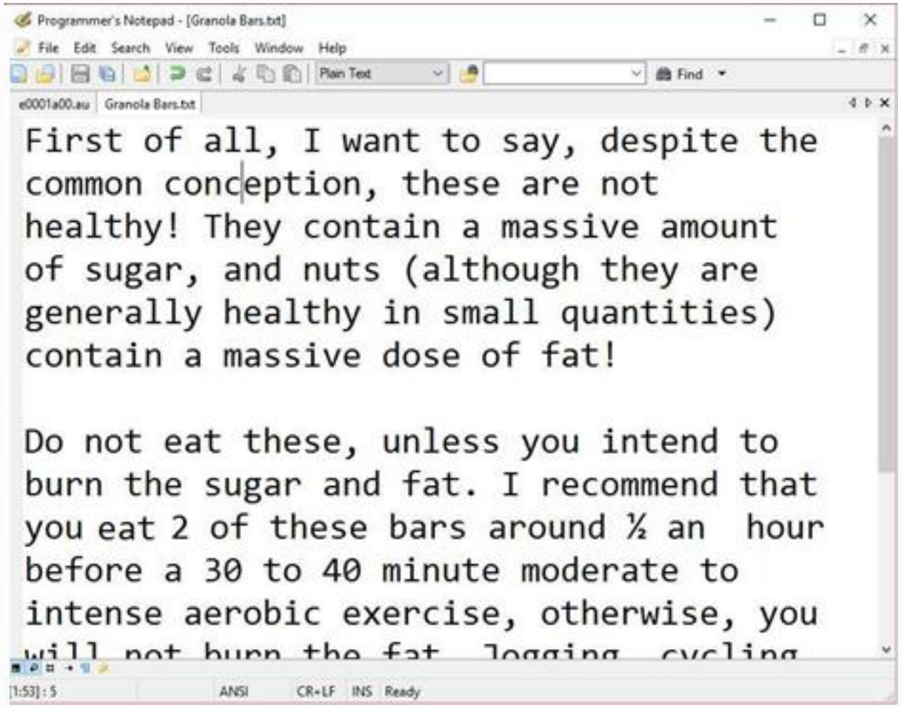
Basics of File Management (7/8)


Programmer's Notepad - [e0001a00.au]
File Edit Search View Tools Window Help
Plain Text
Find
e0001a00.au
Granola Bars.txt
[5:64] : 3071
ANSI
LF
INS
Ready

```
dns.\0NULNULÿÿÿÿACKNULNULNULD¬NULNULSOHNU
¿NULDø>À¨z>NULtÒ¾NUL”«>VTVG>NUL<¸¾NULSTXBE
BELDC3>NULè·¾NULxEOT?2²{>NULŒ½¾NUL1ô>&DC2‹
?@gu>NULDLE†¾NUL`°=*}û=NULXF¾NULÜÍ>p %>NUL
%?îÙ‹>NULú¿NUL@}?èü#?NUL€*¿NUL>6?EOT¥Ÿ>NU
ˆ>NULú¿NUL.~?|GS?NUL¬ù¾NULÌSOH? å†>NULjn¿
?nts>NUL„Ã¾NULÖDC3?ðNAK‡>NULDá¾NULŒDC2?iÕ–
¿NUL¼Æ>ENO¸u>NULFF©¾NUL¸—>.e+>NULÜó¾NULhEN
>EFt>NUL,ENO¿NULDC4#?ï¾œ>NULpÌ¾NULÌî>ìNAK[
ìÇ>NULªm¿NULP´>Êtš>NULÒ1¿NUL,q?ßóº>NUL–3¿N
?=Ë¦>NUL°  ¿NUL”GS?t–’>NULÆ)¿NULþ*?Àƒ¡>NUL€
Š>NUL8FS¿NUL  Â>Õ‡“>NULï¾NUL1¢>Ÿ;)>NUL–GS¿N
```

Binary File

Text File


Programmer's Notepad - [Granola Bars.txt]
File Edit Search View Tools Window Help
Plain Text
Find
e0001a00.au
Granola Bars.txt
[1:53] : 5
ANSI
CR+LF
INS
Ready

```
First of all, I want to say, despite the
common conception, these are not
healthy! They contain a massive amount
of sugar, and nuts (although they are
generally healthy in small quantities)
contain a massive dose of fat!

Do not eat these, unless you intend to
burn the sugar and fat. I recommend that
you eat 2 of these bars around ½ an  hour
before a 30 to 40 minute moderate to
intense aerobic exercise, otherwise, you
will not burn the fat. Jogging, cycling
```

Basics of File Management (8/8)

Text File Stream Vs. Binary File Stream

Text Stream	Binary Stream
Write out separately each of the pieces of data about a given record	Write a whole record data to the file at once
Readable by an editor	Not readable by editor directly
The **main way of opening, reading & writing** is using the file stream object with ✓ *Insertion operator (>>) for reading* ✓ *Extraction operator (<<) for writing*	The **main way of opening, reading & writing** is using the file stream object with ✓ ***write() function** to write to the file the entire record* ✓ ***Read() function** to read from a file a whole record*
It is the default file format	*Require the **ios::binary** offset (file mode) to open the file for binary read and write*

File Manipulation Process (1/13)

- Before actually start using and manipulating files, it is important to discuss the steps to be followed in order to process files.
- Here are the steps that need to be followed in order to perform successful file operations though your program
  - *Declaration of file stream object*
  - Opening a file (attach file stream with a specific file)
  - Check for success opining of the file (optional)
  - Perform File Read/Write Operations
  - Closing a file

Note

- *To perform file read and write (I/O), it is mandatory to include the header file **fstream.h** which defines several classes and functions.*

File Manipulation Process (2/13)

(a) Declaration of File Stream Object

- In order to process file through program, **logical file** must be created on the RAM.
- This logical file is nothing but an **having file data type**.
- **Syntax: object**

  File_Stream File_Stream_Object;

  - Where the "**File_Stream**" refers to either of the three file stream classes *ifstream, ofstream, fstream.*
  - On the other hand the **"File_Stream_Object"** is referring to any valid identifier
- Once the **file stream object** is created, you can use it to open a file and perform various writing and reading operation.

File Manipulation Process (3/13)

(b) Opening a file

- In order to perform read and write operations on the file through a program, the file should be opened.
- Opening a file attaches the stream link to a specific file.
- **Syntax:**

  File_Stream_Object.Open("FileName", File modes) ;

  Where

  - **"File Name" -** *refers to the name of file on which we are going to perform file operations. It should provided with it's full path (where the file is located) and file extension*
  - **"File modes"** – *specify the for what purpose the file is opened and it can be a single mode or multiple file modes.*

Note - the File stream object creation and opening a file can be merged and performed in a single statement (steps), see next slide.

File Manipulation Process (4/13)

File Modes

These are file attributes (modes) for the various kinds of file opening operation

Mode	Description
ios::app	Write all output to the end of the file
ios::ate	✓ Open a file for output and move to the end of the file (normally used to append data to a file). ✓ Data can be written anywhere in the file.
ios::binary	Cause the file to be opened in binary mode.
ios::in	Open a file for input
ios::out	Open a file for output
ios::nocreate	If the file does not exist, the open operation fails.
ios::noreplace	If the file exists, the open operation fails.
ios:trunc	✓ Discard the file's content if it exists ✓ This is also the default action of ios::out
Ios::beg	From beginning of the file
Ios::end	From the end of the file
Ios::curr	From the current file pointer position

File Manipulation Process (5/13)

Example 1: Stream Object Creation and file opening

- Declaring appropriate file stream

```
fstream file;     //file stream object both for reading and writing
```

- Opening a file using stream object with open() function

```
file.open (“myfile.txt”,  ios::in | ios::out);
//opening a file for both reading and writing
```

Example 2: opening a file without using open() function

- The above two statement can be merged and the file can be opened while declaring the stream object

```
fstream file (“myfile.txt”,  ios::in | ios::out)
```

Note: *If the source code file and the file you are operating are raised in the same folder the “file name” is enough to open the file.*

File Manipulation Process (6/13)

(c) Check for success of file opening

- **When opening a file**, especially opening a file for reading, is it critical to **test** for **whether** the **open** operation succeeded or not.
- Two alternatives are available to check for success of file opening
- **Alternative 1:** comparing stream object with NULL

```
ifstream transaction(“sales.txt”);
if (transcation == NULL){
        cout<<“unable to open a file”;
        exit(1);
}
else {
  cout << "\nFile successfully open.\n";
 //read or write operation codes
}
```

File Manipulation Process (7/13)

- **Alternative 2:** using is_open() function with stream object

```
ifstream myfile(“student.txt”);
if (myfile.is_open()){
        cout << "\nFile successfully open.\n";
       //read or write operation codes
}
else {
      cout<<“unable to open a file”;
      exit(1);
}
```

File Manipulation Process (8/13)

(d) Write to and Read from a file

(1) Write to a file

- ➢ The file should opened in reading mode (ios::out)
- ➢ ostream and ofstream are by default use the ios::out mode

- **How to write to a file?** Using the following three alternatives
- **Alternative 1:** Using the Output stream object in combination with << (insertion operator)
  - ✓ E.g. ofstream Inf; Inf<<variable; Inf<<"Hello, World);
- **Alternative 2:** Using the write() function
  - ✓ **Syntax:** streamObject.write(string data, size)
  - ✓ E.g. ifstream Inf; Inf.write ("Hello, World", 10);
- **Alternative 3:** Using the put() function
  - ✓ Used to write one character only
  - ✓ **Syntax:** inf.put(character);

File Manipulation Process (9/13)

(2) Read from a file

- ➢ The file should opened in reading mode (ios::out)
- ➢ istream and ifstream are by default use the ios::out mode

- **How to read from a file?** Using the following three alternatives

- **Alternative 1:** Using the Input stream object in combination with >> (extraction operator)
  - ✓ E.g. ifstream Inf; Inf>>variable;

- **Alternative 2:** Using the read() function
  - ✓ **Syntax:** streamObject.read(string data, size)
  - ✓ E.g. ifstream Inf; Inf.read (“Hello, World”, 10);

- **Alternative 3:** Using the get() or getline() function
  - ✓ Can be used to read one character and one line string respectively
  - ✓ **Syntax:** inf.get(character); or
    inf.get(string data, size); or inf.getline(string data, size);

File Manipulation Process (10/13)

Note:

- fstream is by default use the ios::out and ios::in mode
- read() and write() functions are a **binary function** which used to write or read an object or record (sequence of bytes) to/from a binary file mainly.
- To write to and/or read from a binary, the file should opened in a *binary file mode (ios::binary)*

```
fstream file;
file.open("file.dat", ios::out | ios::binary);
```

- Multiple file modes can be used by ORing them together

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc );
```

File Manipulation Process (11/13)

(e) Closing a file

- ***Why we need to close a file?***
  - To make the stream object to be free and can be used with more than one file.
  - To avoid a logical error that will occur because of not closing opened file
- ***How to close a file? Using two possible methods***
  - *Using close() function*

    stream-object.close();

    e.g. inf.close();
  - *Using clear() function*

    stream-object.clear();

    e.g. inf.clear();
- *In both cases the file that opened using the **"inf"** file stream will be closed*

File Manipulation Process (12/13)

The prototype of file read and write functions

Function	Prototype and Syntax (1st and 2nd line respectively)

1. get()	istream & get(char & ch);
	infile.get(ch);

	istream &get (char *buf, int num, char delim = '\n'); Infile.get((char*)&buf, sizeof(buf));

2. Put()	ostream & put (char ch);
	outfile.put(ch);

3. getline()	istream & getline (char *buf, int num, char delim ='\n'); Infile.getline((char*)&buf, sizeof(buf));

4. read()	istream & read (unsigned char * buf, int num);
	Infile.read((char*)&buf, sizeof(buf));

5. write()	ostream & write (const unsigned char * buf, int num);
	outfile.write((char*)&buf, sizeof(buf));

Note: *infile – input file stream, outfile - output file stream, ch – character variable, buffer – string variable*

File Manipulation Process (13/13)

Description of the functions

Function	Description
1. get()	Read a *character or one line string* from the associated stream and puts the value in *ch or buff*, and returns a reference to the stream
2. Put()	Write a character stored on *ch* to the stream and returns a reference to the stream.
3. getline()	Read a *one line string* from the associated stream and puts the value in *buff*, and returns a reference to the stream
4. read()	✓ Reads *num bytes* from the associated stream, and puts them in a *memory buffer* (pointed to by buf). ✓ If the end-of-file is reached before num characters have been read, then read () stops and puts the read character on the memory buffer
5. write()	Writes *num bytes* to the associated stream from the *memory buffer* (pointed to by buf).

Stream state functions (1/2)

- The stream state member functions give the information status like end of the file has been reached or file open failure and so on.

Function	Description
bad()	✓ Returns true if a reading or writing operation fails. ✓ For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.
fail()	Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.
eof()	Returns true if a file open for reading has reached the end.
good()	It is the most generic state flag: It returns false in the same cases in which calling any of the previous functions would return true.
is_open()	It used to check either the file is opened or not.
Peek()	Used to obtain next character in the input stream without removing it from that stream
Ignore()	Reads and discards characters until either num characters have been ignored
putback ()	Operates in the reveres of peek()

Stream state functions (2/2)

File stream state offsets

- *eofbit* - 1 when end-of-file is encountered; 0 otherwise
- *Openbit* - 1 when a is opened successfully; 0 otherwise
- *failbit* - 1 when a (possibly) nonfatal I/O error has occurred; 0 otherwise
- *badbit* - 1 when a fatal I/O error has occurred; 0 otherwise

For example

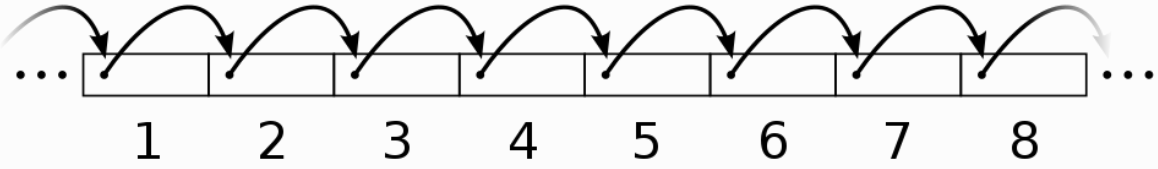
```
#include<fstream.h>
int main()
{
            Ifstream f1;
            f1.open(“sam”);
            while(!f1.eof()){
             // reading file
            }
}
```

File Random Access (1/5)

- Basically there are two types of file access: sequential and random.

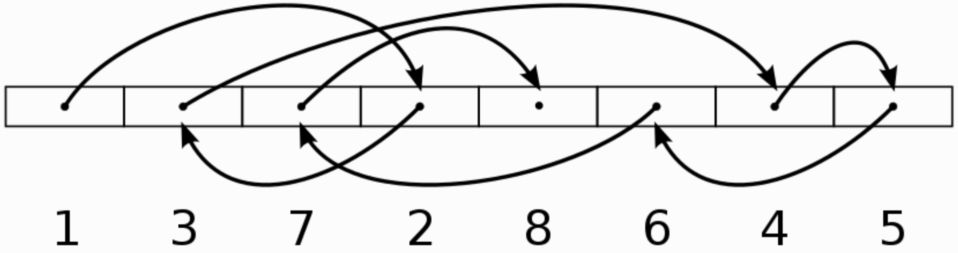
***(a) Sequential access*.**

- ➢ With this type of file access one must read the data in order, much like with a tape, whether the data is really stored on tape or not.



(b) Random access (or *direct access*).

- ➢ This type of file access lets you jump to any location in the file, then to any other, etc., all in a reasonable amount of time.



File Random Access (2/5)

File Pointers

- Each file object has two integer values associated with it
  - ***get pointer***
  - ***put pointer***
- The values of the pointers specify the byte number in the file where reading or writing will take place.

- By default
  - **reading pointer** is set at the ***beginning***
  - **writing pointer** is set at the **end** (when the file open is opened in *ios::app or ios::ate* mode)

File Random Access (3/5)

Random Access

- Allow you to read from and write to an arbitrary location in the file.
- The **seekg()** and **tellg()** functions allow you to set and examine the **get pointer** where as the **seekp()** and **tellp()** functions allow you to set and examine the **put pointer**.

(1) tellg() and tellp()

- *Does not has any parameter*
- *Used to get the current file pointer position*
- *tellg() – returns the current pointer position for reading*
- *tellp() – returns the current pointer position for writing*

File Random Access (4/5)

(2) seekg() and seekp()

- *Used to set (specify) the file pointer position for reading and writing respectively*
- ***Syntax:***

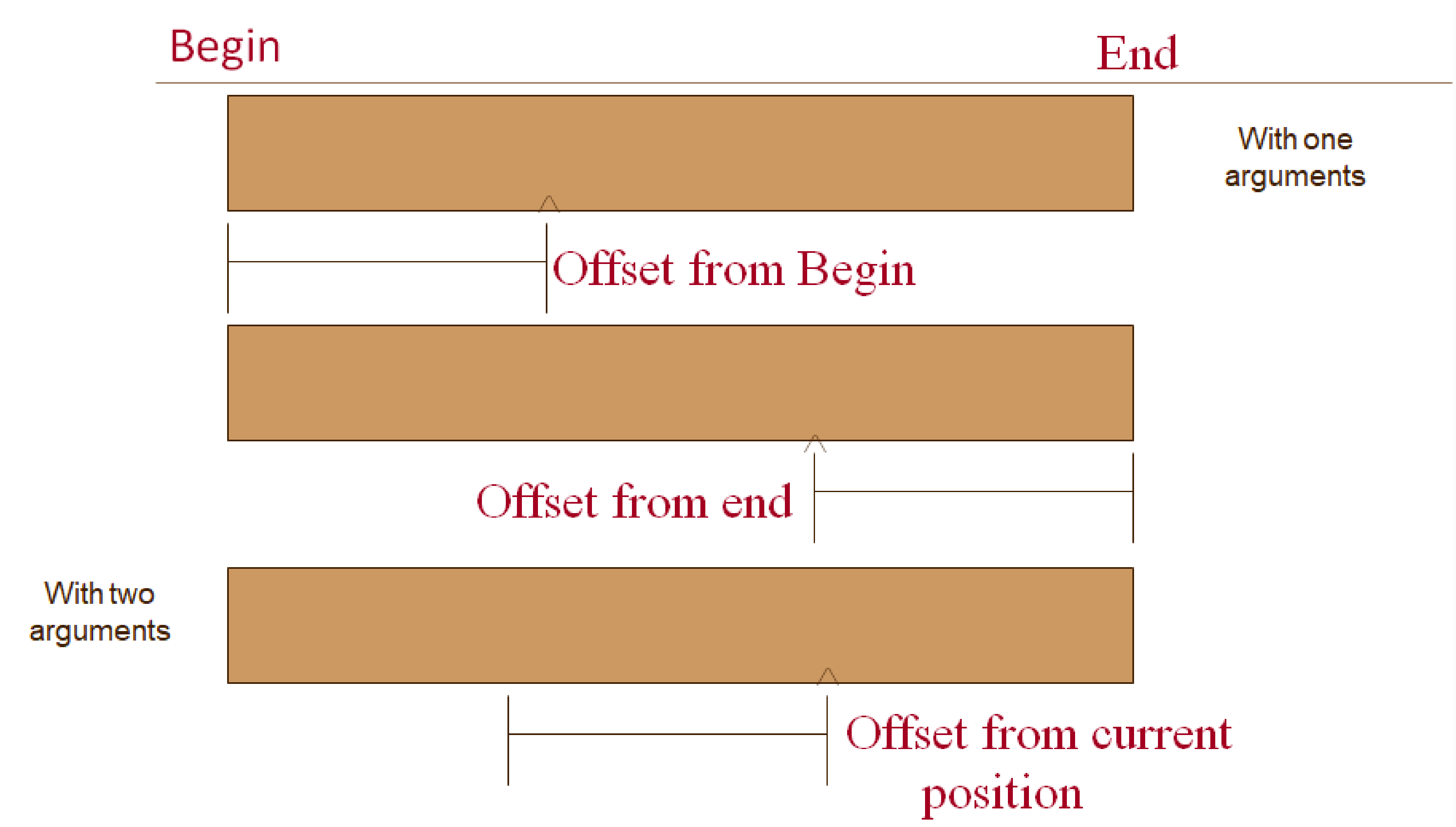
```
stream-object.seekg(size in byte, offset)
stream-object.seekp(size in byte, offset)
```

- *E.g.*

```
fstream inf;
inf.open(“myfile.txt”, ios::in | ios::out);
inf.seekg(24, ios::beg);   //move get pointer 24 bytes from beginning
inf.seekp(-8, ios::end);   //move put pointer 8 bytes from end of file
inf.seekg(4, ios::curr);   //move get pointer 4 bytes from current position
```

File Random Access (5/5)

File pointer offsets



Practical Examples (1/12)

Example 1a: write to a text file

```
// Create a sequential file.
#include <iostream>
#include <fstream>
using namespace std;

int main(){
        ofstream outClientFile( "clients.txt", ios::out );     //creating ofstream object
                                                               //and opening a file
    // check if unable to create file
     if ( !outClientFile ) {
           cout << "File could not be opened" << endl;
            exit( 1 );
        } // end if
int account;
char name[ 30 ], ch=‘y’;
double balance;
```

Practical Examples (2/12)

Example 1a (cont’d)

```
// read account, name and balance from cin, then place in file
cout << "Enter the account, name, and balance separate by space." << endl;
cout<< "Enter \’N\’ to end input.\n? ";
while (ch == ‘y’)
{
     cin >> account >> name >> balance;
     outClientFile << account << ' ' << name << ' ' << balance<< endl;
      cout << "? ";
       cin>>ch;
} // end while

   outClientFile.close();      // // close ofstream file

  return 0;
} // end main
```

Practical Examples (3/12)

Example 1b: read from a text file

```
// Create a sequential file.
#include <iostream>
#include <fstream>
using namespace std;

int main(){
        ifstream inClientFile( "clients.txt", ios::in);          //creating ifstream object and
                                                                 //opening a file
    // check if unable to create file
     if ( !inClientFile.is_open() ) {
           cout << "File could not be opened" << endl;
            exit( 1 );
        } // end if
int account;
char name[ 30 ];
double balance;
```

Practical Examples (4/12)

Example 1b (cont’d)

```
// read account, name and balance from cin, then place in file
cout << “The User bank account details\n”;
cout<< “Account \t Name\t  Balance\n ";

while (inClientFile.eof() == false)
{
      inClientFile >> account >> name >> balance;
      cout<< account << ‘\t' << name << ‘\t' << balance<< endl;
} // end while

    inClientFile.close();        // close ifstream file

   return 0;
} // end main
```

Practical Examples (5/12)

Example 2: get() and put() functions

```
#include <iostream>
#include <fstream>
using namespace std;

int main(){
        char str[80], c, d, ans;
        ofstream outfl("try.txt");
// read a string from keyboard and write to a file.
do{
               cout<<"please give the string : ";
               gets(str);              outfl<<str;
               cout <<"do you want to write more...<y/n> : ";  ans=getch();
    }while(ans=='y');
    outfl<<'\0';
    outfl.close();
```

Practical Examples (6/12)

Example 2 (cont’d)

```
// copying file content using get() and put() functions
    Ifstream infl("try.txt");
    ofstream out("cod.dat");

    cout <<"reading from created file and copying to other file\n";
    infl.get(c);
    do{
         d=c+1;
        cout<<c<<d<<'\n';
        out.put(d);
        infl.get(c);
    }while (c!='\0');
    out<<'\0';

    infl.close();
    out.close();
    }
```

Practical Examples (7/12)

Example 4a: write to binary file

```
#include <iostream>
#include <fstream>
using namespace std;
struct Student{
            int roll;
            char name[25];
            float marks;
      } stud;
void getdata(){
     cout<<"\n\nEnter Roll : ";
     cin>>stud.roll;
     cout<<"\nEnter Name : ";
     cin>>stud.name;
     cout<<"\nEnter Marks : ";
     cin>>stud.marks;
}
```

```
void AddRecord(){
     fstream outf;
     outf.open("Student.dat",ios::app|ios::binary);

     getdata();
     outf.write( (char *) &stud, sizeof(stud) );
     outf.close();
}
int main()
 {
   char ch='n';
   do{
        AddRecord();
       cout<<"\nwant to add more (y/n) : ";
       get(ch);
       } while(ch=='y' || ch=='Y');
        cout<<"\nData written successfully...";
   }
```

Practical Examples (8/12)

Example 4b: read from binary file

```
#include <iostream>
#include <fstream>
using namespace std;
struct Student{
            int roll;
            char name[25];
            float marks;
      } stud;

void putData()
{
    cout<<"\n”<<stud.Roll;
    cout<<“\t”<<stud.Name;
    cout<<“\t”<<stud.Marks;
}
```

```
void Display(){
     fstream inf;
     inf.open("Student.dat",ios::in|ios::binary);

     cout<<"\n\tRoll\tName\tMarks\n";

     inf.read( (char *) &Stu, sizeof(stud) );
     while(inf != NULL){
             putData();
             inf.read( (char *) &Stu, sizeof(stud) );
     }
     inf.close();
}

int main() {
      Display ();
   }
```

Practical Examples (9/12)

Example 5: random access

```
/* C++ File Pointers and Random Access
* This program demonstrates the concept
 * of file pointers and random access */
#include <iostream>
#include <fstream>
using namespace std;
#include <string.h>

struct Student{
            int roll;
            char name[25];
            float marks;
              char grade;
      } stud1, stud;
```

```
void getData(){
    cout<<“Student Inof:\n”;
    cout<<"Rollno: ";   cin>>stud1.rollno;
    cout<<"Name: ";    cin>>stud1.name;
    cout<<"Marks: ";   cin>>stud1.marks;

    float marks = stud1.marks;
    if(marks>=75)  {stud1.grade = 'A'; }
    else if(marks>=60){stud1.grade = 'B'; }
    else if(marks>=50){stud1.grade = 'C'; }
    else if(marks>=40){stud1.grade = 'D'; }
    else{  stud1.grade = 'F'; }
}

int getrno(){
            return stud1.rollno;
  }
```

Practical Examples (10/12)

Example 5 (cont’d)

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


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

cout<<"Enter new details.\n";
char nam[20]=" ";
float mks;
```

```
cout<<"New name:(Enter '.' to retain z old): ";
cin>>nam;
cout<<"New marks:(Press -1 to retain z old):";
cin>>mks;

if(strcmp(nam, ".")!=0){
                strcpy(name, nam);
    }
if(mks != -1){
    stud1.marks = mks;
    if(stud1.marks>=75){stud1.grade = 'A';}
    else if(stud1.marks>=60){stud1.grade = 'B';}
    else if(marks>=50){stud1.grade = 'C';}
    else if(stud1.marks>=40){stud1.grade = 'D';}
    else{ grade = 'F';              }
     }
}
```

Practical Examples (11/12)

Example 5 (cont’d)

```
int main()
{
   fstream fio("marks.dat", ios::in | ios::out);
   char ans='y‘;
   while(ans=='y' || ans=='Y')
   {
       getdata();
       fio.write((char *)&stud1, sizeof(stud1));
       cout<<"Record added to the file\n";
       cout<<"\nWant to enter more ? (y/n)..";
       cin>>ans;
    }

//search and modify a record on the file
      int rno;     long pos;      char found='f';
}
```

```
cout<<"Enter rollno of student
whose record is to be modified: ";
cin>>rno;

fio.seekg(0);
int size = sizeof(stud1);
while(!fio.eof()){
     pos = fio.tellg();
     fio.read((char *)&stud1, size);
     if(getrno() == rno){
         modify();
         fio.seekg(pos);
         fio.write((char *)&stud1, size);
         found = 't';
         break;
         }
}
```

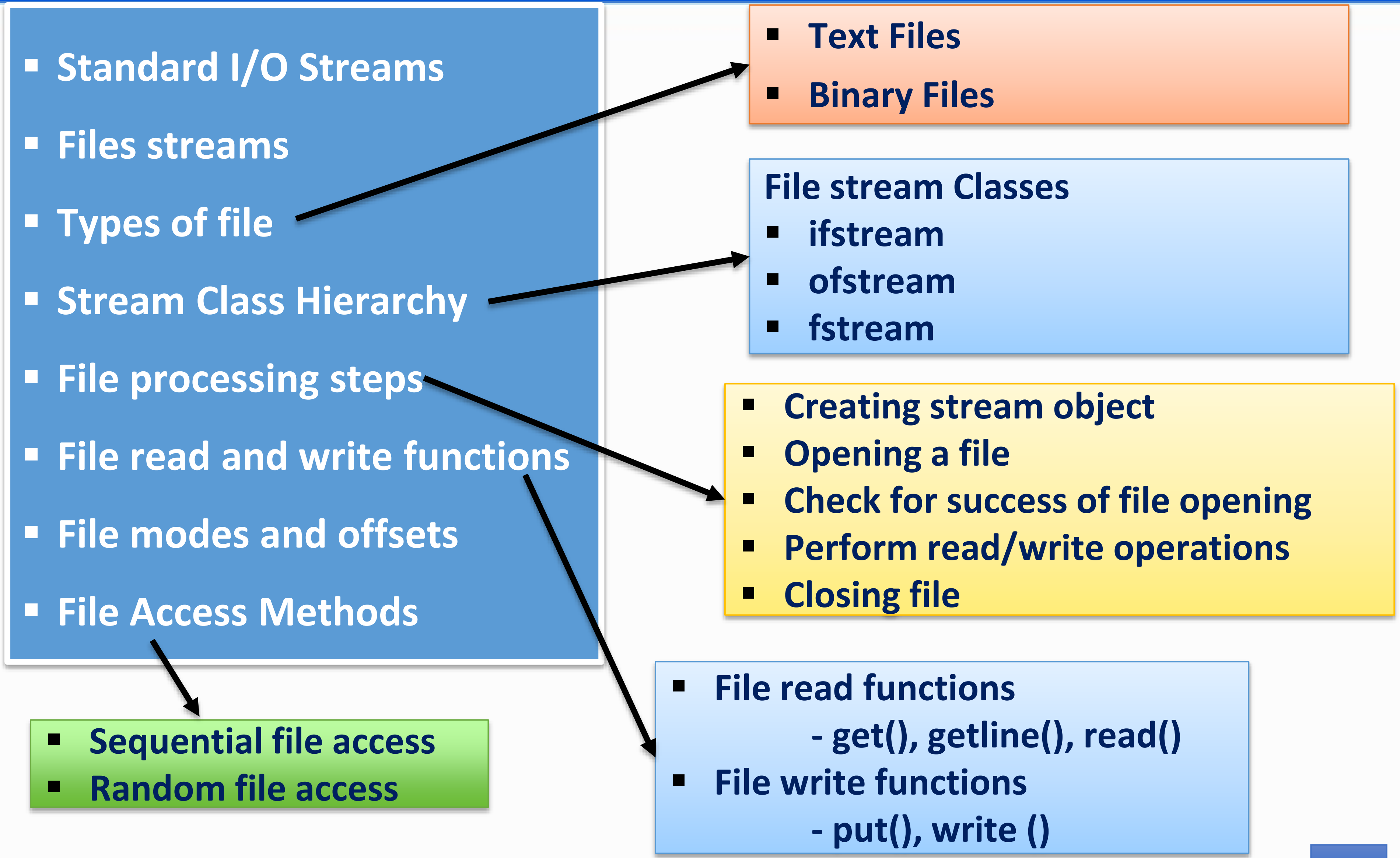
Practical Examples (12/12)

Example 5 (cont’d)

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

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

Summary



- Standard I/O Streams
- Files streams
- Types of file
- Stream Class Hierarchy
- File processing steps
- File read and write functions
- File modes and offsets
- File Access Methods

- Text Files
- Binary Files

File stream Classes

- ifstream
- ofstream
- fstream

- Creating stream object
- Opening a file
- Check for success of file opening
- Perform read/write operations
- Closing file

- File read functions
  - get(), getline(), read()
- File write functions
  - put(), write ()

- Sequential file access
- Random file access

MCQ

1. What is meant by ofstream in C++ ?

    (a) Reads from a file　　(b) Writes to a file

    (c) All of the above　　(d) None of the Above

2. How many types of output stream classes are there in C++ ?

    (a) 2　(b) 3　(c) 1　(d) none

3. Which function is used to position back from the end of file object ?

    (a) seekp　　(b) seekg

    (c) tellp　　(d) tellg

4. Which header file is used for reading and writing to a file ?

    (a) #include<file>　　(b) #include<iostream>

    (c) #include<fstream>　　(d) None of the Above

5. Which one is always faster in writing on C++ ?

    (a) Reading from the network　　(b) Writing to a file

    (c) Writing to memory　　(d) None of the Above

MCQ

6. Which will be used with physical devices to interact from C++ program ?

(a) Streams (b) Programs

(c) Library (d) None of the Above

7. Which header files is required for creating and reading data files ?

(a) console.h (b) ifstream.h

(c) ofstream.h (d) fstream.h

8. What is the benefit of C++ input and output over C input and output ?

(a) Exception (b) Type safety

(c) All of the above (d) None of the Above

9. By default, all the files in C++ are opened in _________ mode.

(a) Text (b) Binary

(c) ASCII d) None

10. Which is the default mode of the opening using the fstream class?

(a) ios::in b) ios::out

(c) ios::in|ios::out (d) ios::trunc

Practical Exercise

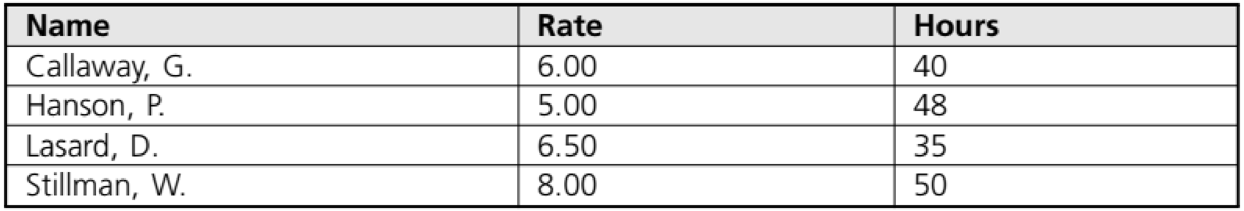
1. Write a program that accept N student record from the keyboard & store the list on a file “D:\\ Test.txt” in a text file format. Also write an other program that reads students record from the text file “D:|\ Test.txt” and display on the screen. (tip: create a header file which contain definition of two function getRecord() and displayrecord() and include it in your program).
2. Modify your program in Q1 to store the student records a binary file.
3. Write a program which prints a table listing the number of occurrences of the lower-case characters 'a' to 'z' in a file "Sheet5Ex5.cpp". Declare only one variable of type "ifstream", one variable of type "char", and two variables of type "int". The program should produce output such as the following

```
CHARACTER          OCCURRENCES
    a                  38
    b                  5
    c                  35
    -                  -
    -                  -
```

Practical Exercise

4. Write a function that takes the name of a file (char*) that contains integer records, an array of int and the address of a variable count. Define the function to read the file into the array. Assume that the array has enough space to hold the file. count should be updated to the number of entries in the file.

5. Create a text file containing the following data (without the headings)



Name	Rate	Hours
Callaway, G.	6.00	40
Hanson, P.	5.00	48
Lasard, D.	6.50	35
Stillman, W.	8.00	50

Write a C++ program that uses the information in the file created to produce the following pay report for each employee:

Name Pay Rate Hours Regular Pay Overtime Pay Gross-Pay

Compute regular pay as any hours worked up to and including 40 hours multiplied by the pay rate. Compute overtime pay as any hours worked above 40 hours times a pay rate of 1.5 multiplied by the regular rate. The gross pay is the sum of regular and overtime pay. At the end of the report, the program should display the totals of the regular, overtime, and gross pay columns

Practical Exercise

6. **(Search)** A bank's customer records are to be stored in a file and read into a set of arrays so that a customer's record can be accessed randomly by account number. Create the file by entering five customer records, with each record consisting of an integer account number (starting with account number 1000), a first name (maximum of 10 characters), a last name (maximum of 15 characters), and a double-precision number for the account balance. After the file is created, write a C++ program that requests a user-input account number and displays the corresponding name and account balance from the file.

7. Write a C++ program that permits users to enter the following information about your small company's 10 employees, sorts the information in ascending ID number, and then writes the sorted information to a file:

ID No. Sex(M/F) Hourly-Wage Years-with-the-Company

After the records are stored successfully,

(a) write a program that reads the file created one record at a time, asks for the number of hours each employee worked each month, and calculates and displays each employee's total pay for the month

(b) Develop a program that reads the file created and changes the hourly wage or years for each employee, and creates a new updated file

Reading Resources/Materials

Chapter 14:

- ✓ **P. Deitel , H. Deitel**; C++ how to program [10th edition], Global Edition (2017)

Chapter 14:

- ✓ Diane Zak; An Introduction to Programming with C++ (8th Edition), 2016 Cengage Learning

Chapter 6:

- ✓ Walter Savitch; Problem Solving With C++ [10th edition, University of California, San Diego, 2018

Chapter 18:

- ✓ Herbert Schildt; C++   From the Ground Up (3rd Edition), 2003 McGraw-Hill, California 94710 U.S.A.

Thank You
For Your Attention!!