CE142:OBJECT ORIENTED PROGRAMMING WITH C++

December 2018 – May 2019

UNIT 12

Working with Files

Devang Patel Institute of Advance Technology and Research

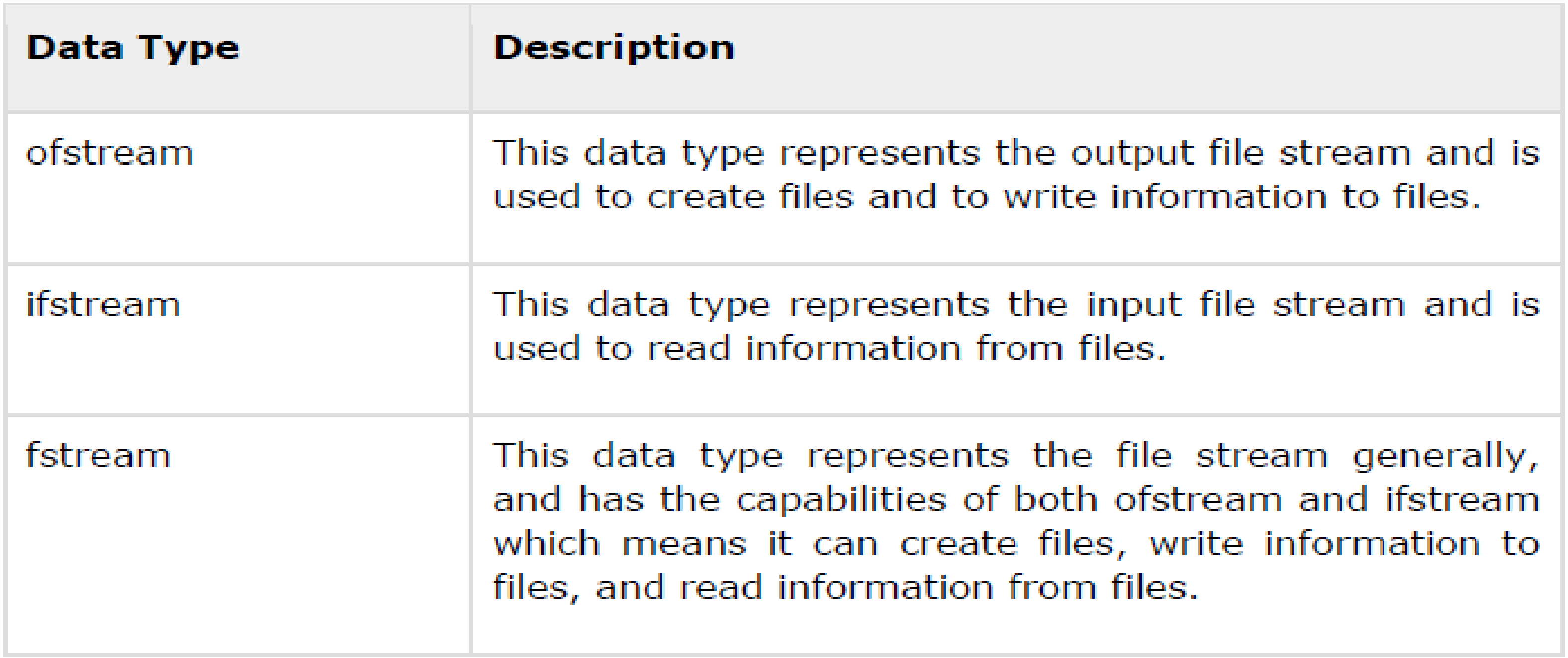
Content

Continued…

- File Pointers
- Error Handling
- Command line arguments

Classes for File Stream Operators

- To read and write from a file requires another standard C++ library called fstream, which defines three new data types:



Data Type	Description
ofstream	This data type represents the output file stream and is used to create files and to write information to files.
ifstream	This data type represents the input file stream and is used to read information from files.
fstream	This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

Opening a File

- A file must be opened before you can read from it or write to it.
- Either **ofstream** or **fstream** object may be used to open a file for writing.
- And **ifstream** object is used to open a file for reading purpose only.
- Two ways to open a file:
  - Constructor
  - Using the open() function

Opening a File using a Constructor

```
ofstream outfile(“hello.txt”);
```

- Here outfile is an object of type ofstream which is associated with the file hello.txt and is used to open it.
- Then just like using cin and cout objects…

```
outfile << “The Total is:”;
outfile << sum;
```

Opening a File using Open() function

```
ofstream obj;
obj.open("hello.txt");
```

- The only difference is using a function called open to open the file instead of using the constructor of the ofstream class.
- Similarly ifstream and fstream objects can also be associated with files.
- If you want to open two files at the same time, use two different objects. SIMPLE!

Closing a File

- When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files.
- But it is always a good practice that a programmer should close all the opened files before program termination.

```
fileobj.close();
```

Detecting End of File

- Detection of EOF condition is necessary for preventing any further attempt to read data from the file.

```
If(fileobj.eof() != 0)
        cout<<"End of File!"; //means obj.eof() is non-zero
Else
        cour<<"Not the end of file";
```

Another way…

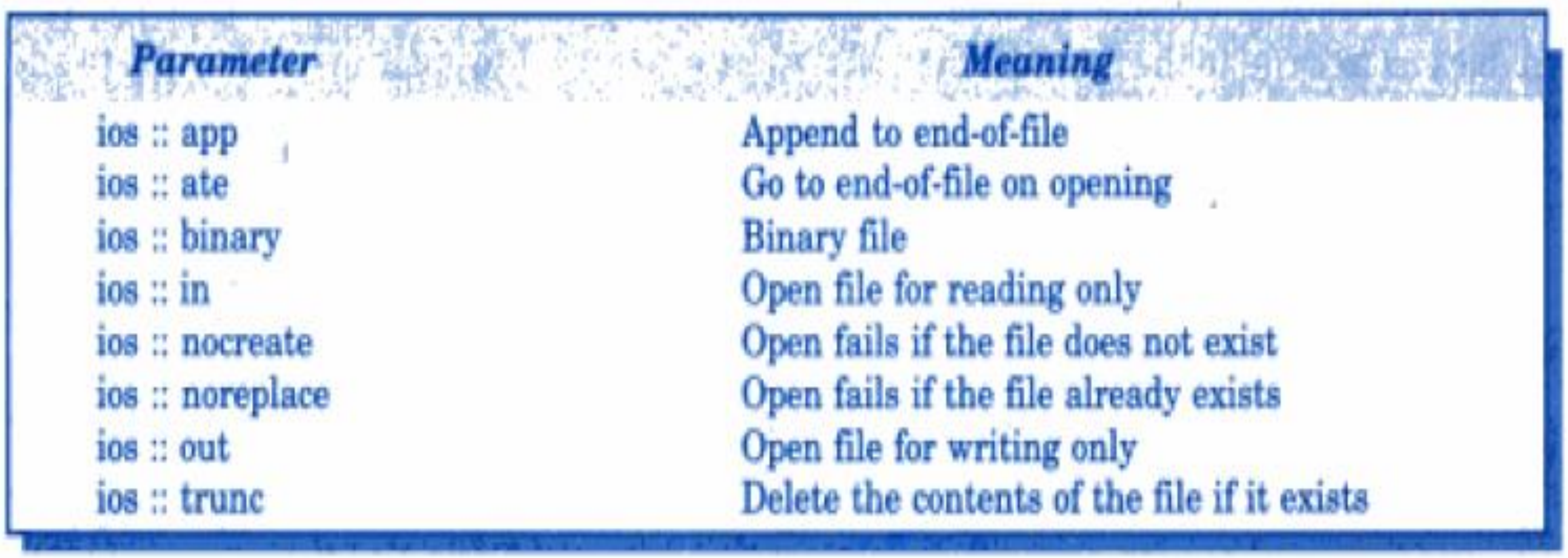
```
If(fileobj)
        cout<<"Everything's fine!";
Else
        cout<<"Some error occurred with the file object";

        //A 0 value returned by the file object may also
indicate other errors besides End-of-file
        //So better to use fobj.eof() function
```

File Modes

- In the open() function to open files you can also specify the mode in which you want to open the file.

fileobj.open(“filename.txt”, mode);



Parameter	*Meaning*
ios :: app	Append to end-of-file
ios :: ate	Go to end-of-file on opening
ios :: binary	Binary file
ios :: in	Open file for reading only
ios :: nocreate	Open fails if the file does not exist
ios :: noreplace	Open fails if the file already exists
ios :: out	Open file for writing only
ios :: trunc	Delete the contents of the file if it exists

- You can combine two modes by using he OR pipe operator |

File Pointers

- Each file has two associated pointers known as file pointers.
- One of them is the input pointer or **get pointer.** It is used for reading the contents of a given file location.
- And the other is called the output pointer or **put pointer**. Its used for writing to a given file location.
- Each time a read or write operation is performed the appropriate pointer is automatically advanced.

Functions for Manipulation of File Pointers

seekg()	Moves get pointer (input) to a specified location.
seekp()	Moves put pointer(output) to a specified location.
tellg()	Gives the current position of the get pointer.
tellp()	Gives the current position of the put pointer.

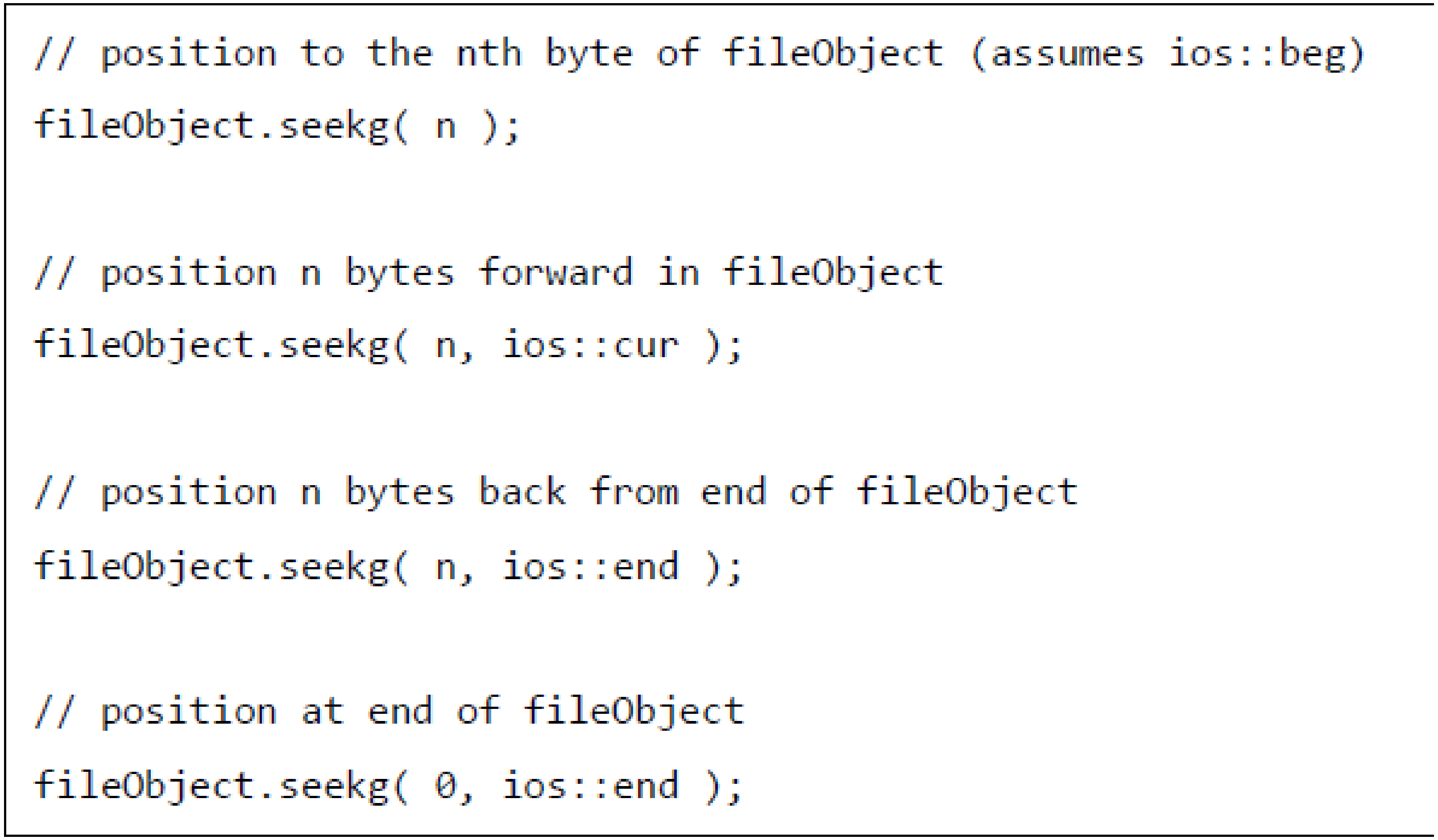
```
seekg (offset, refposition);
seekp (offset, refposition);
```

OFFSET IS IN BYTES

ios::beg	start of the file
ios::cur	current position of the pointer
ios::end	End of the file

FLAGS FOR POSITIONS

Functions for Manipulation of File Pointers



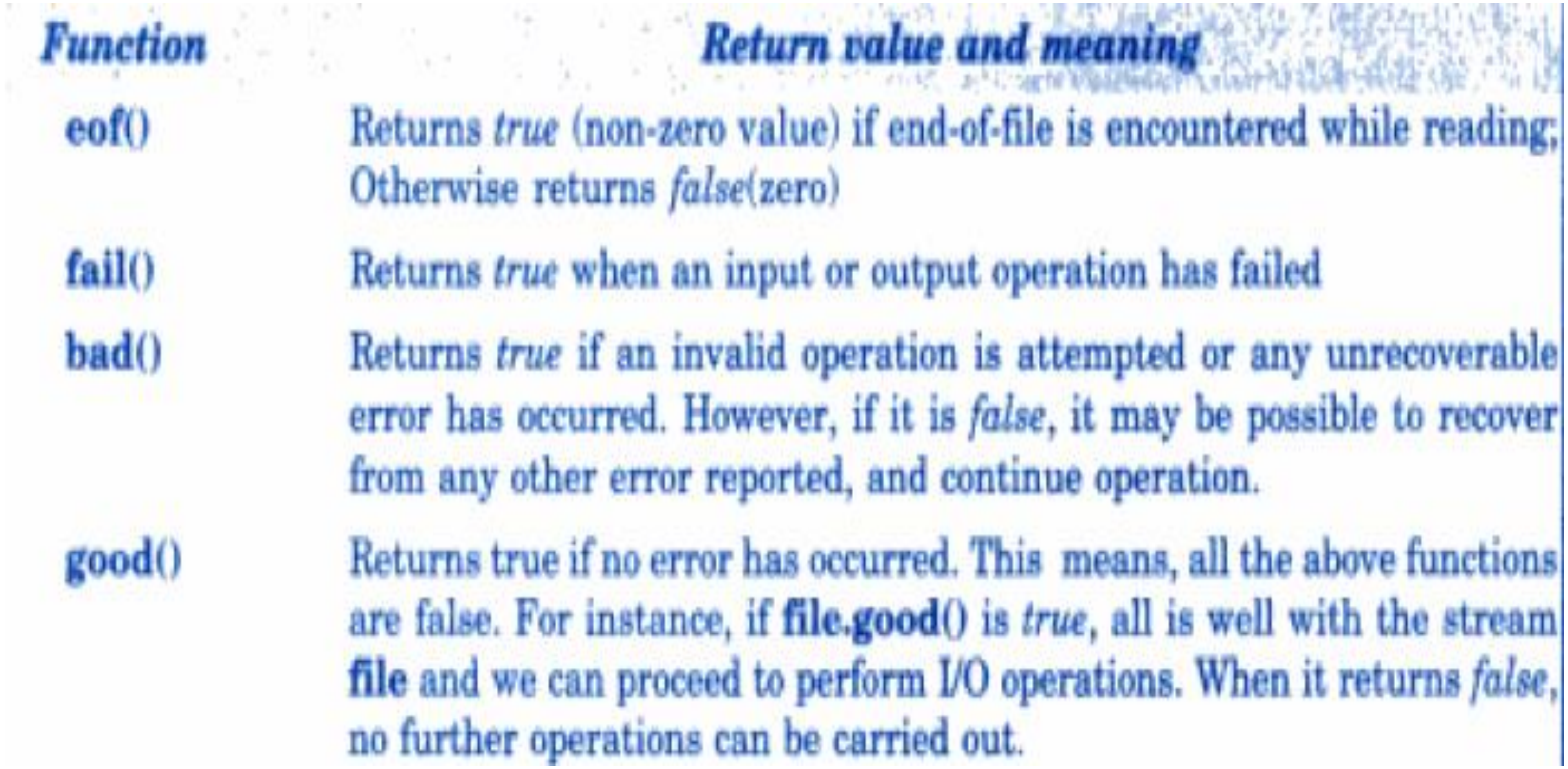
```
// position to the nth byte of fileObject (assumes ios::beg)
fileObject.seekg( n );

// position n bytes forward in fileObject
fileObject.seekg( n, ios::cur );

// position n bytes back from end of fileObject
fileObject.seekg( n, ios::end );

// position at end of fileObject
fileObject.seekg( 0, ios::end );
```

File Error Handling Functions (To be used with file stream object)



Function	Return value and meaning
eof()	Returns *true* (non-zero value) if end-of-file is encountered while reading; Otherwise returns *false*(zero)
fail()	Returns *true* when an input or output operation has failed
bad()	Returns *true* if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is *false*, it may be possible to recover from any other error reported, and continue operation.
good()	Returns true if no error has occurred. This means, all the above functions are false. For instance, if **file.good()** is *true*, all is well with the stream **file** and we can proceed to perform I/O operations. When it returns *false*, no further operations can be carried out.

Command Line Arguments

- Command-line arguments are used when invoking(running) a program from the command prompt/ terminal.
- To read command-line arguments, the main() function must itself be given two arguments: **argc** and **argv**.

```
//helloworld.cpp
int main(int argc, char* argv[] )
…
…

//Running the program from command prompt with 3 command line arguments: uno, dos and tres

C://helloworld 3 uno dos tres
```

Command Line Arguments

- The first, argc (for *argument count*), represents the total number of command-line arguments and then we have a character pointer array called argv each element of which stores a string as an argument
- The command-line arguments are those typed by the user; they are delimited by the space character.
- In the previous example, uno, dos and tres are stored in argv and 3 is stored in argc

Presentation Prepared By:

Mr. Phenil Buch

Contact us:

phenilbuch.ce@charusat.ac.in
dweepnagarg.ce@charusat.ac.in
parthgoel.ce@charusat.ac.in
hardikjayswal.it@charusat.ac.in
dipakramoliya.ce@charusat.ac.in
krishnapatel.ce@charusat.ac.in
khushipatel.ce@charusat.ac.in

Subject Teachers:

Ms. Dweepna Garg
Subject Coordinator

Mr. Parth Goel
https://parthgoelblog.wordpress.com

Mr. Hardik Jayswal

Ms. Khushi Patel

Thank you!