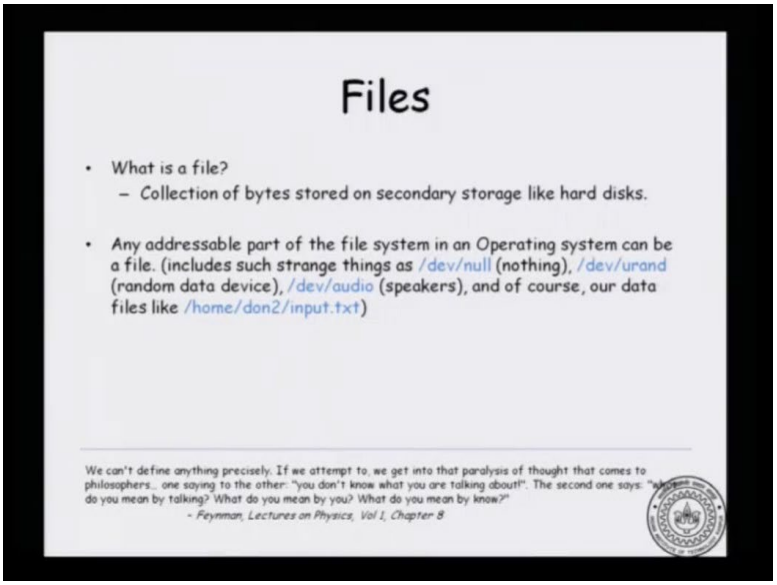
Introduction to Programming in C
Department of Computer Science and Engineering

In this video, we will look at a few basic things about file handling in C. This is a vast topic in itself and we will see just the basics of this. So, let us begin by describing what are files.

(Refer Slide Time: 00:15)



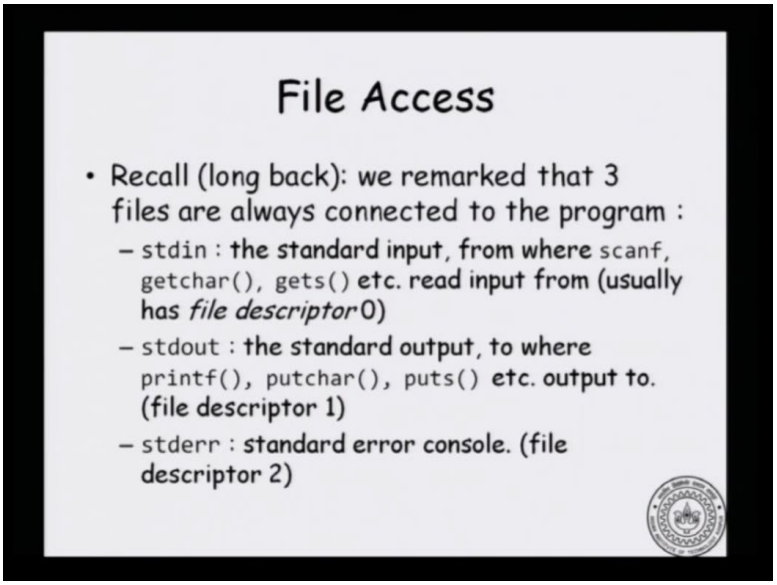
Now, you would think that the most natural way to define what a file is, it is a set of bytes or a collection of bytes sitting in secondary storage, like may be your hard drive, may be your CD ROM drive or DVD drive or something, it is known as a secondary storage device. But, the actual description is that, any addressable part of the file system in an operating system is a file. Now, this includes extremely bizarre strange things.

For example, /dev/null in Linux, this stands for nothing. So, if you write to /dev/null, it is like discarding the data. Similarly, /dev/urand, this is the random data device. If you read from here, you will get random data, /dev/audio is speakers. So, if you write some data into that, it will be heard on the speakers. And of course, plain old data files, for example, in a home directory you may have /home/don 2/input.text.

So, input.text is just a collection of bytes. So, we will not bother with defining what a files is, but it is something that can be manipulated using the file system interface. So,

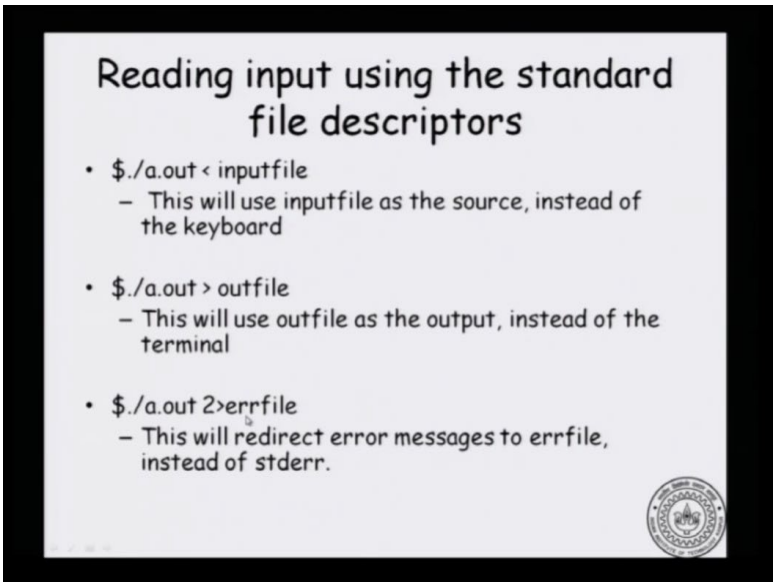
you can open a file system interface to that file, you can read from that file, you can write into that file, you can close that file and so on.

(Refer Slide Time: 01:53)



Now, recall that in one of our earlier lectures, I said that there are three files works which are available by default to all c programs. So, these are standard input stdin. This is associated usually with a key board and this is where scanf, getchar, gets this kinds of functions get their input from, it has file descriptor 0. Similarly, standard output stdout is where printf, putchar, puts all these functions output the data to. This is usually visible on the terminal under screen, this has file descriptor 1. We also have a third file, which is known as standard error. This is the standard error console and which has file descriptor 2. Usually, you can print error messages to stderr. We have not seen how to print error messages to stderr, so far we will see that in this video.

(Refer Slide Time: 03:00)

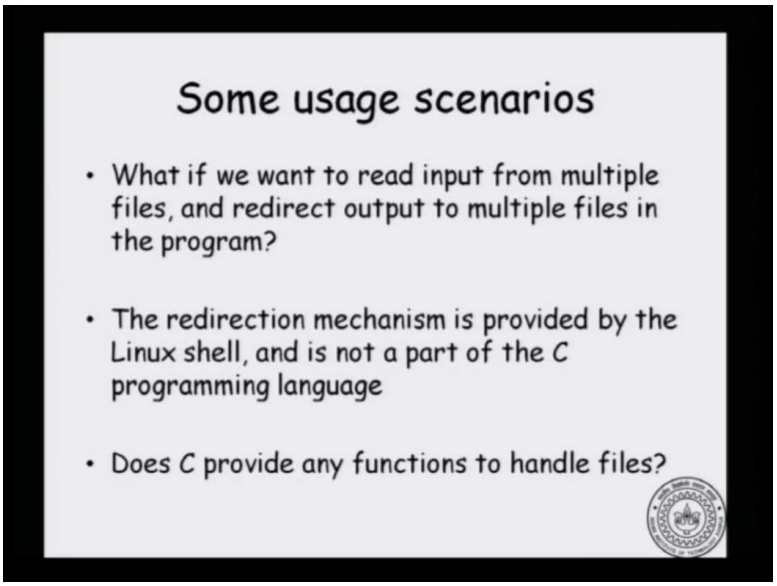


Now, how do you read input using standard file descriptors, but from sources other than key board and so on. So, for example if you are running a.out and you want to take the input not from the key board, but, from an input file. You can say /a.out < input file, this says that the input is coming from the file, input file. So, this is the input file as the source, instead of the key board.

Similarly, if you want to redirect, so if you want to redirect the output to a particular output file, instead of the screen, you can says /a.out > out file. So, this will use the out file as the output instead of the terminal. And if you want to output something the error messages to error file, you can say /a.out 2 > error file, 2 stands for the standard error console. So, if you say redirect this to this file, you will say ok.

The standard output should get the standard output messages and the standard error messages should go to err file.

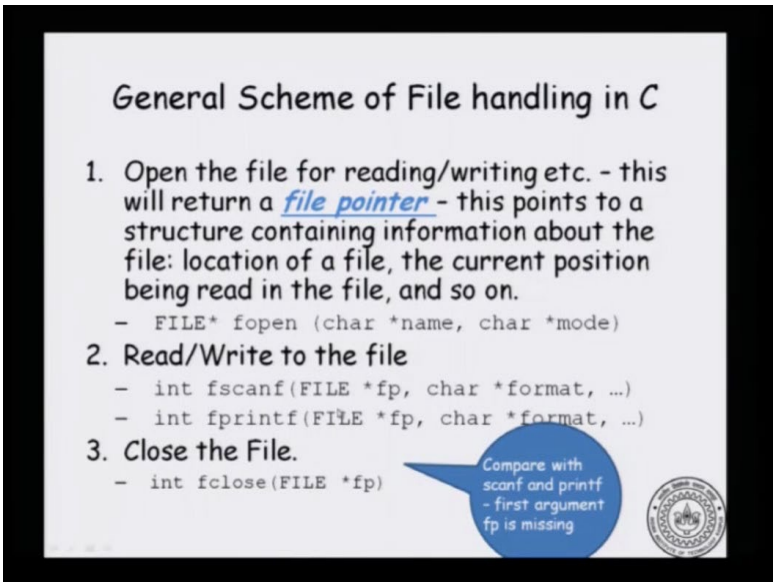
(Refer Slide Time: 04:17)



So, Linux gives you some facilities to take input from other files using the standard input and the output. So, what you say is that, instead of the standard input, you can use this <, > arrows, in order to redirect input from some file or output to another file or error to another file. So, this is the facility that Linux gives you. But, consider the general situation, when you have a program, you want to read the input from multiple files and may be output to multiple files.

So, this is the general situation, we just saw how to take input from one particular input file, how to output to another output file using the redirection operator, the < and > operation on Linux? So, the redirection mechanism is provided by the Linux shell and is not a part of the C program in language. So, is there a way to do it in C itself, rather than using the facilities of Linux. So, can we read from other files, other than the standard key board? Can we write into other files, other than writing on to the screen, standard output and so on.

(Refer Slide Time: 05:30)



So, we will look at the general scheme of file handling in C, all these functions that I am going to talk about are in stdio.h, itself. So, you do not need to include any more files. So, if you want to open the file for reading or writing etcetera, we need to first open the file. The three standard files stdout, stdin and stderr are available to the program. Any other file, you have to open the file. And the function to do that is fopen takes two arguments name and mode, we will see what these are soon and it returns something called a file pointer.

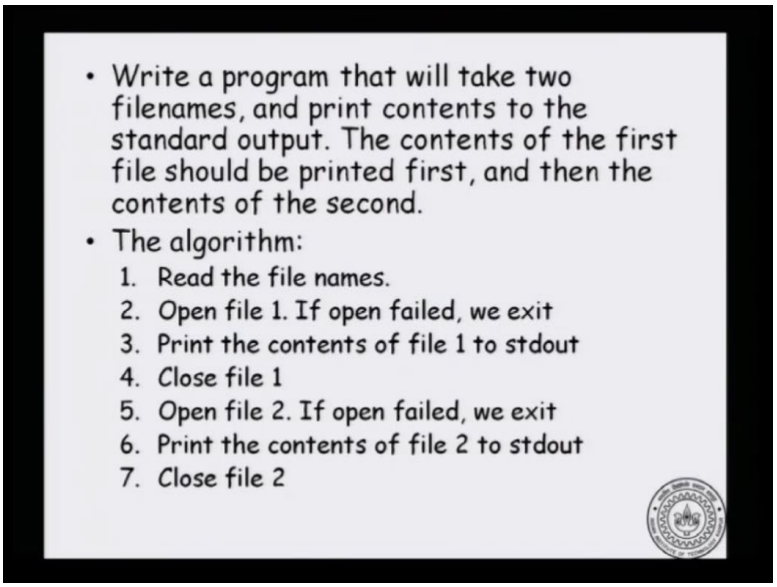
A file pointer is a pointer to a structure and that structure contains a lot of information about the file. For example, where is it situated, the current position being read in the file. So, may be you are read 1000 bytes and you are about to read the 1000 and first byte. So, it has that information and various maintenance information, about the file. Now, in order to read or write into the file, you can use fscanf or fprintf. These are the analogs of scanf and printf, which allow you to right to an arbitrary files.

It takes three arguments, at least two arguments, the first is the file pointer, where you want to write the file, where you want to read from the file and so on. And then, there is a format specified, just as a normal scanf or normal printf and then further arguments. So, the difference here is that, whereas scanf and printf started with a format specifies, we have an additional file pointer in the beginning.

So, compare with a scanf and the printf, the first argument fp is missing. This is because, scanf just assumes that the file it has to read from is this standard input. And printf assumes that it has to print to the standard output and to close the file, you say fclosefp. And notice the way, the fscanf, fprintf and fclose work, they do not take the file name as input, only fopen takes the name of the file as input.

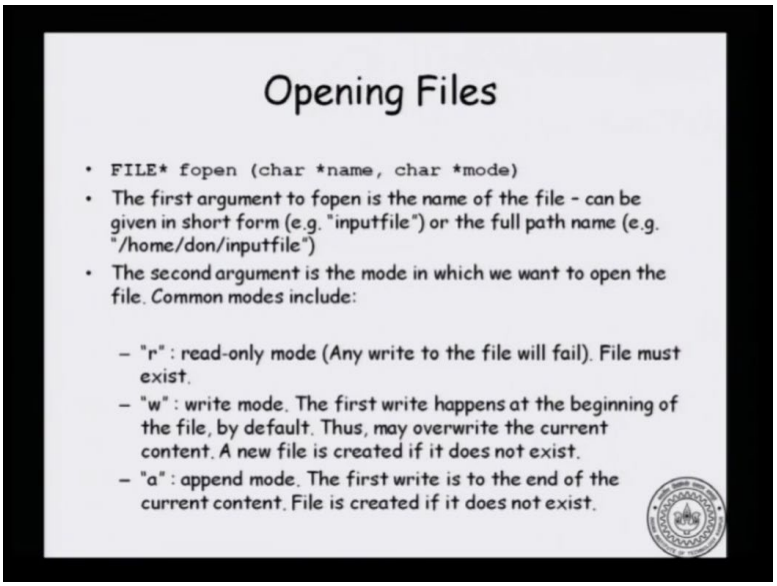
Whatever fopen returns the file pointer, those are the arguments to fscanf, fprintf and fclose. This is because, once a file has been opened, all the information that fscanf, fprintf and fclose need are already in the structure pointer 2 by fp.

(Refer Slide Time: 08:18)



Now, let us write a very simple program, it takes the names of two files and what it does is, it first prints the contents of the first file and then prints the contents of the second file and these will be output to the standard output. What is the algorithm? It is very simple, you have to first read the file names, then open file 1, if open fails, we exit. Now, you print the contents of file 1 to stdout, after you have done, you close file 1. Then, you open file 2, check whether open has succeeded, if it has failed, we exit. Then, print the contents of file 2 to stdout close file 2 and that is it.

(Refer Slide Time: 09:08)



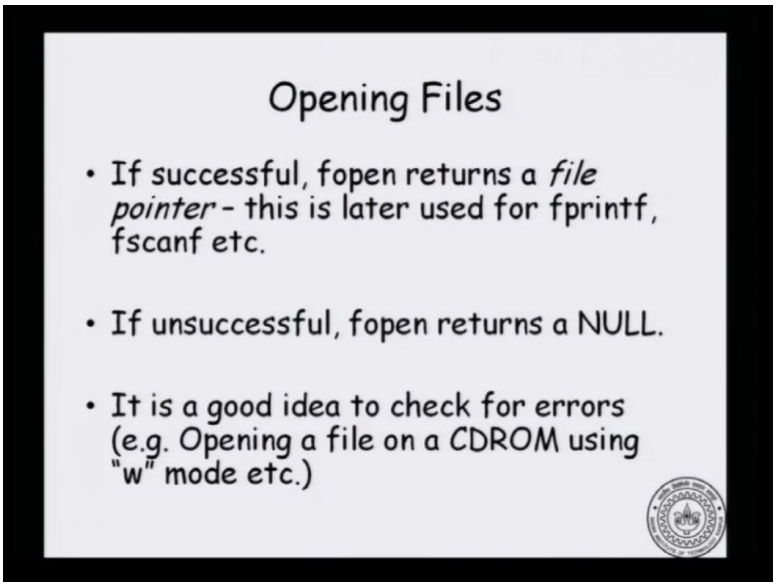
So, let us see what each of these steps in slightly more detail. How do you open the file? We open it using a standard call called fopen, fopen takes two arguments, the name and the mode as character pointers and returns a file point. The first argument name is the name of the file and the name of the file can be given in short form. Suppose, you are already in a directory, where that file is situated. Then, you can just give the name of the file. For example, input file or you can give the full path name of that file in the operating systems.

So, for example input file may be in the directory /home/don. So, in that case you can give the name as /home/don/input file. So, this will be the full path file, either of this is accepted. Now, the second argument is the mode, this is the way in which you want to open the file. So, what are the common modes? For example, if you give r, this will open the file in read only mode. This is, if you want to just read a file and not write to that file.

There are also other situations, where the medium itself may not support writing. For example, if you have a CD ROM disk then you cannot write to that. So, it can only be opened in a read only mode, if you give w, this is the write mode. Now, the first write happens at the beginning of the file. So, if the file already exists, it will be over written. If a file does not exist, so this is the name of a new file that we support commonly is known as the append mode, we specify that by saying the mode is a.

So, if you open the file for append mode, then instead of writing at the first location of the file, it will write at the end of the current file. So, if the file does not exist, then it will start from the first location. If the file exists, it will go to the end of the file and start writing from there. So, append does not overwrite the file.

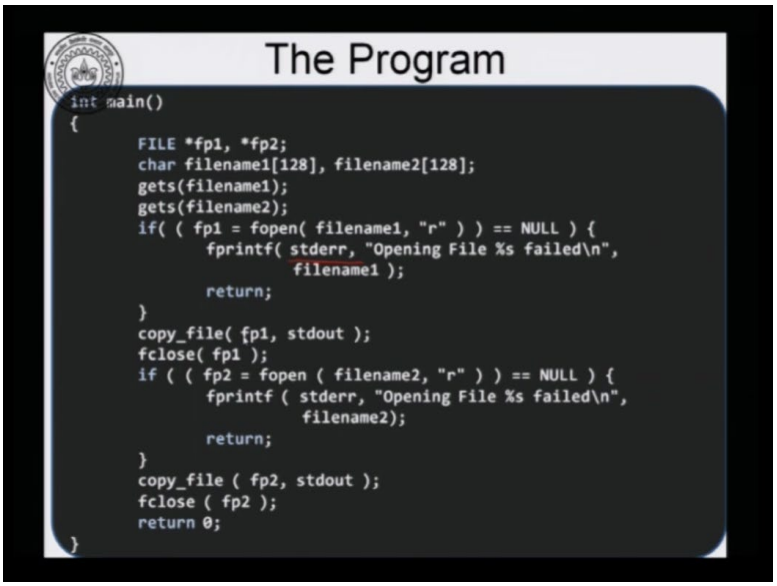
(Refer Slide Time: 11:38)



We have seen the arguments of fopen. Now, let us look at what it returns? If successful fopen returns what is known as a file pointer. This is later used for fprintf, fscanf, fclose as I just mentioned. If unsuccessful, the file may be you try to open a nonexistence file for reading or you try to write to a file which cannot be return to. For example, it is a file sitting inside a CD ROM drive and you are not allowed to write to it. So, if you try to open the file in write mode, then you have a problem.

So, for whatever reason if the file open does not succeed, then the fopen returns in null and it is always a good idea to check for these errors. So, just try opening a file and always check whether it has return the null.

(Refer Slide Time: 12:33)



So, let us write the program that we were discussing, which will take two input files and print one file and then print the other file. So, the program is fairly simple, we have a main function, we have two file pointers *fp1 and *fp2. And then, two file names filename1 and filename2, you get the filename1 from the input, you get filename2 from input using gets functions. Now, what we have to first do is, write the contents of the first file.

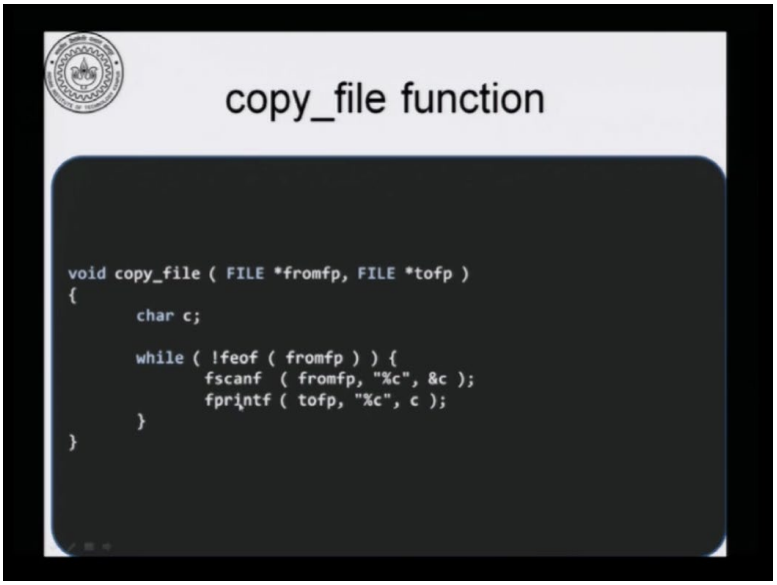
So, try opening the file, so if (fp1 = fopen(filename1, “r”)). Because, we just need to read from the file, we do not need to write into it. So, open it in r mode, if it is successful that is, if it is or rather if it has failed. So, if it has returned a null, then you just say printf that it has failed. And here is for the first time, we have seeing how to print to this standard error. So, stderr is any other file is similar to any other file, you can just say fprintf stderr. And then, opening file failed filename1.

So, we try to open filename with filename1 as the name, but there was some error. So, you print that to the error terminal, which is stderr. Now, once you do that will call the function copy_file(fp1, stdout). So, here is a function that we will write, which will copy from a source file to a destination file and what it takes are pointers to those files.

Once you have done, you close the file 1 and then, you repeat the whole process, the exactly the same process for file 2. So, try to open it, if there is an error, you print the error message to stderr, then copy_file(fp2, stdout) and finally, close the file. Once you

have done, you can return from main. So, now what is left is, what is this copy file function?

(Refer Slide Time: 14:59)



So, let us look at the copy file function. Now, there are two ways to start writing any function which takes files as arguments. One is you can take the file name as the argument itself and within the function, try to open the file. So, you will get a file pointer and you can start reading from the file using fscanf I am writing to the file using fprintf, this is possible.

It is somewhat more convenient to say that I am assume that the files are already open and I am getting the file names as the point using file pointers. So, this avoids duplication of work, the main does not have to open the file. And then, every function has to open the file again and again. Instead, what you can just say that, I assume that the caller function has already is a file open and I will just take a file pointer as the argument.

So, let us look at this function, it is a void function. So, it does not return anything, it just performs an action, it is name is copy file takes two arguments, fromfp which is a file pointer to after to the source file and tofp, which is file pointer to the designation file. And what is a function do? We have a character c and here is a function, we will see in a later video. But, right now it just checks whether fromfp has encountered enter file.

So, feof just tells you whether you have done with the from file. So, if you are not done with a from file what you do is, you scan one character from the from file, so fscanf(fromfp, “%c”, &c). So, this will read one character from the source file fromfp and read it into the variable c. What we have to do is, to print that to tofp. So, you say fprintf (tofp, “%c”, c). So, this is exactly like scanf and printf, but taking one extra argument.

So, in the case of scanf you just says, what is the source file that is the file pointer argument. In the case of fprintf, you have to take the designation file which is tofp, that is the extra argument in that expression.