# Objectives:

1. File I/O
   1. To become familiar with the concept of file I/O
   2. To learn basic guidelines in implementing file I/O functions
   3. To learn how file I/O is implemented

# FILE I/O

Till now, we have focused on taking input from console and printing on console. This was done using standard input output streams provided by iostream. So, for example, the cin we have been using is a standard input stream and cout is the standard output stream. For reading and writing to files, the file stream must be used. For this purpose, we need to include fstream as follows.

#include <fstream>

C++ provides the following classes to perform output and input of characters to/from files:

**ofstream:** Stream class to write on files

**ifstream:** Stream class to read from files

**fstream:** Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes istream and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we must associate these streams with physical files. Let us see an example:

// basic file operations #include <iostream> #include <fstream> using namespace std;

int main ()



{

ofstream myfile;

myfile.open ("example.txt");

myfile << "Writing this to a file.\n"; myfile.close();

}

This code creates a file called example.txt and inserts a sentence into it in the same way we are used to do with cout, but using the file stream myfile instead.

# 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 (i.e., an object 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.

To open a file with a stream object we use its member function open:

open (filename, mode);

Where filename is a string 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, 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. |
| ios::trunc | If the file is opened for output operations and it already existed, its previous  content is deleted and replaced by the new one. |

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open:



ofstream myfile;

myfile.open ("example.bin", ios::out | ios::app | ios::binary);

Each of the open member functions of classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

|  |  |
| --- | --- |
| **class** | **default mode parameter** |
| ofstream | ios::out |
| ifstream | ios::in |
| fstream | ios::in | ios::out |

# CLOSING A FILE

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function close. This member function takes flushes the associated buffers and closes the file:

myfile.close();

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes. In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function close. The following example shows how to read one word from file “input.txt” and output to file “output.txt”.

# EXAMPLE NO 1: SIMPLE FILE INPUT AND OUTPUT

#include <iostream>

#include <fstream> using namespace std; int main()

{

char outmsg[100]; cout<<"Enter Message: ";

cin.getline(outmsg,100);

// Defining a char array

// Take input from the user in char array



char readmsg[50];

// Defining a char array

ofstream fout;

ifstream fin;

// creating object of ofstream

// creating object of ifstream

fout.open("output.txt");

fin.open("input.txt");

// Now open output.txt for writing

// Now open input.txt file for input

fout << outmsg;

// Write string msg onto file

fin >> readmsg;

cout << readmsg<<endl;

// Read from input file

// Print readmsg on screen

fin.close();

fout.close();

// Close files after usage

// Close files after usage

}

For executing example 1, you must already have a text file by the name “input.txt” in the same folder as your code. Assuming the file contains,

Welcome fileio

is easy

the console output of example no 1 will be

Welcome

Note that only the first word in the text file is read by the file stream. This is because all C++ streams by default stop on whitespaces.

# CHECKING FOR FILE OPENING ISSUES

You should never assume that the file IO was successful. For example, if you are opening a file for reading, you should have checks in your code to make sure there is no unexpected behavior if the file is missing. This can be done using the fail() function of the streams. The following example shows how to make such checks.



# EXAMPLE NO 2: CHECKING FOR CORRECT OPENING OF FILE

#include <fstream> #include <iostream> using namespace std; int main()

{

char readmsg[50]; ifstream fin;

// Now open input.txt file for input fin.open("input.txt");

if(!fin.fail())

{

// Read from input file fin >> readmsg;

// Print readmsg on screen cout << readmsg<<endl;

}

fin.close();

}

The above check may also be important for output file stream since there may be errors while opening a file for writing.

**READING FILE TO END**

As discussed earlier, reading file using streams only reads a single word (until first whitespace). If there is no word or we have reached end of file, *fin>>variable* returns False. The following example shows how to read a file until end and print it on screen.

# Example No 3: Reading Complete File and Printing on Screen

#include <iostream> #include <fstream> using namespace std; int main ()

{

char str[50]; ifstream fin; int count = 0;



fin.open("input.txt");

if(!fin.fail())

{

while(fin>>str) // Continue reading until end

{

cout<<str; count++;

}

cout<<endl<<count<<endl; // Print number of times file was read

}

fin.close();

}

**Output**

Welcomefileioiseasy 4

The above example will read the file “input.txt” until end and print it on screen. It also shows how many times the file was read.

# -----------------------------------------x-----------------------------------------x-----------------------------------------

**LAB TASKS**

**Task No 1: File Write**

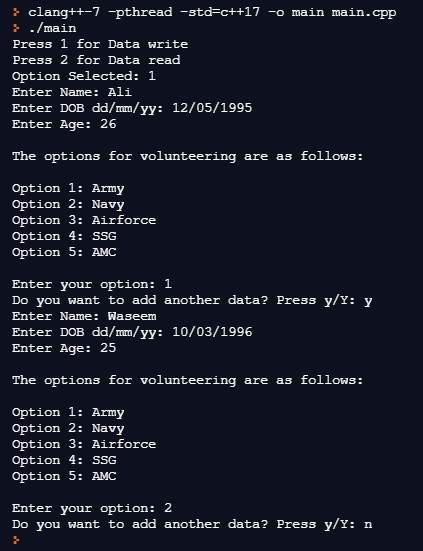
Armed forces recruiting agency is collecting data of volunteers for its five main branches i.e., ARMY, NAVY, AIRFORCE, SSG and AMC. You are required to write a C++ program that collects data of the volunteers and store them in a file. For each volunteer, following information needs to be stored in a file (all details are in character array form).

* Name
* DOB (dd/mm/yyyy)
* Age

The program will read the preferences from a file “branches.txt” and display the options in the screen for the user to select. The user may select the preference by pressing 1 for Army, 2 for Navy etc. Once the user has entered details of the volunteer i.e., name, DOB, age, and preference, the details of each volunteer is stored in a file. The program will ask the user if you want to enter another data if YES then another set of details will be entered and the information will be appended in the output file i.e., “*data.txt*”. if NO then the program will terminate.

* *Use Do While to check if the user wants to add another data*
* *Define a char space[2]={' ','\0'}; and append it at the end of each volunteer data.*
* *Use fout.open("data.txt", ios::app) to append the next user information (ios::app appends the content to the current content of the file.)*
* *You need to take input in a character array only*

# REQUIRED OUTPUT



**Task No 2: File Read**

Write a C++ program that reads the entire contents of the file i.e., “*data.txt*” and display the entries on the screen.

* The program will use file I/O stream to display the contents of “*data.txt*” file.
* You can either write a separate program or use IF/ELSE for either write or read data in task 1.

# REQUIRED OUTPUT