

Object Oriented Programming (CT-260)

Lab 12

Introduction to Filing

Objectives

The objective of this lab is to familiarize students with filing operations in C++. By the end of this lab, students will be able to write data to a file, and reading data from a file along with closing a file.

Tools Required

DevC++ IDE / Visual Studio / Visual Code

Course Coordinator –

Course Instructor –

Lab Instructor –

Prepared By Department of Computer Science and Information Technology

NED University of Engineering and Technology

File Handling in C++

File handling in C++ is a mechanism to store the output of a program in a file and help perform various operations on it. Files help store these data permanently on a storage device.

For achieving file handling, we need to follow the following steps:

STEP 1-Creating a file

STEP 2-Opening a file

STEP 3-Writing data into the file

STEP 4-Reading data from the file

STEP 5-Closing a file.

Streams in C++

We give input to the executing program and the execution program gives back the output. The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called stream. In other words, streams are nothing but the flow of data in a sequence.

The input and output operation between the executing program and the devices like keyboard and monitor are known as “console I/O operation”. The input and output operation between the executing program and files are known as “disk I/O operation”.

In **input operations**, the bytes flow from a device (e.g., a keyboard, a disk drive, a network connection, etc.) to main memory.

In **output operations**, bytes flow from main memory to a device (e.g., a display screen, a printer, a disk drive, a network connection, etc.).

So far, we have been using the iostream standard library, which provides **cin** and **cout** methods for reading from standard input and writing to standard output respectively. C++ provides you with a library that comes with methods for file handling.

```
#include <iostream> // contains cin and cout
```

Stream Input/Output Classes and Objects

The I/O system of C++ contains a set of classes which define the file handling methods. These include ifstream, ofstream and fstream classes. These classes are derived from fstream and from the corresponding iostream class. These classes, designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.

fstream library:

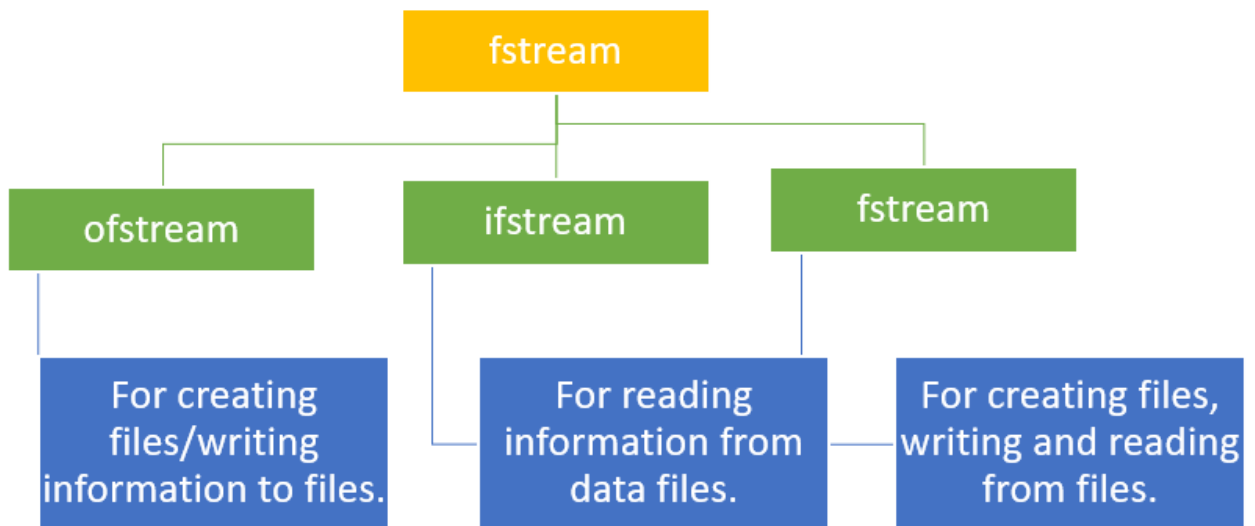
Three classes known as ofstream, ifstream, and fstream are utilised by the C++ fstream library to manage files.

1. **ofstream:** This class assists in creating and adding data to the file that is created using the program's output. The output stream is another name for it.
2. **ifstream:** This class, sometimes referred to as the input stream, is used to read data from files.
3. **fstream:** Ofstream and ifstream are combined into one class in this case. It offers the ability to write, read, and create files.

You must include the fstream as a header file, similar to how we define iostream in the header, in order to use the following classes.

```
#include<fstream>
```

Stream-I/O template hierarchy



C++ File Operations

For managing files, C++ has four main operations.

- ✓ **open()**: Create a file using the open() method.
- ✓ **read()**: The data from the file is read using the read() function.
- ✓ **write()**: Write new data to a file using the write() function.
- ✓ **close()**: The file is closed using the close() method.

1. Opening a File

We must first open a file in order to read or write data to it. We can open file by

1. passing file name in constructor at the time of object creation
2. using the open method

File creation and opening using constructor method

We can open file using a constructor like this:

```

ifstream (const char* filename, ios_base::openmode mode= ios_base::in);
ifstream fin(filename, openmode) by default openmode = ios::in
ifstream fin("filename");
  
```

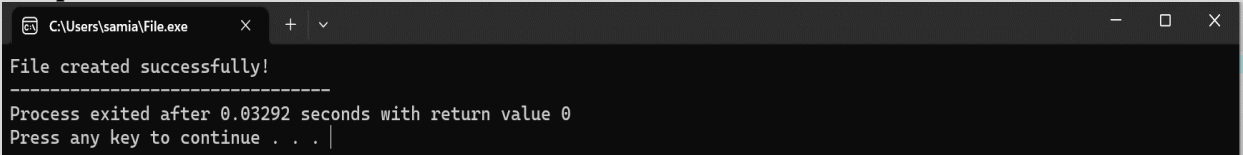
Example: File creation

```

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

int main(){
    ofstream my_file ("file1");
    if (!my_file){
        cout << "File not created!";
    }
    else {
        cout << "File created successfully!";
        my_file.close();
    }
}

```

Output:


```

C:\Users\samia\File.exe
File created successfully!
-----
Process exited after 0.03292 seconds with return value 0
Press any key to continue . . .

```

File open using open() method

The three objects, that is, fstream, ofstream, and ifstream, have the open() function defined in them. The function takes this syntax:

```

// Calling of default constructor
ifstream fin;
fin.open(filename, openmode)
fin.open("filename");

```

The filename denotes the name of the file whereas, mode determines different modes to open the file.

Mode	Description
ios::in	File opened in reading mode
ios::out	File opened in writing mode
ios::app	File opened in append mode
ios::ate	File opened in reading mode but read and write performed at the end of the file
ios::binary	File opened in binary mode
ios::trunc	File opened in truncate mode
ios::nocreate	The file opens only if it exists
ios::noreplace	The file opens only if it doesn't exist

These Mode Flags help you open file in a mode of your choice as per the need of the program. If you want to set more than one open mode, just use the OR operator | like:

```
ios::ate | ios::binary
```

Example: Creating a file using open ()

```

#include <iostream>
#include <fstream>

```

```
using namespace std;

int main(){
    fstream my_file;
    my_file.open("file2", ios::out);
    if (!my_file){
        cout << "File not created!";
    }
    else {
        cout << "File created successfully!";
        my_file.close();
    }
    return 0;
}
```

Opening a file associates a file stream variable declared in the program with a physical file at the source, such as a disk. In the case of an input file:

- ✓ the file must exist before the open statement executes.
- ✓ If the file does not exist, the open statement fails and the input stream enters the fail state

An output file does not have to exist before it is opened, if the output file does not exist, the computer prepares an empty file for output. If the designated output file already exists, by default, the old contents are erased when the file is opened.

File validation before access

Before accessing a file in C++, it's a good practice to validate its existence and ensure that you have the necessary permissions to access it. We can use the predefined Boolean functions like `.good()` and `.open()` to check if a file is available and the connection is open.

Example: Validating file existence and access

```
#include <iostream>
#include <fstream>

bool fileExists(const std::string& filePath) {
    std::ifstream file(filePath);
    return file.good();
}

bool canAccessFile(const std::string& filePath) {
    std::ifstream file(filePath);
    return file.is_open();
}

int main() {
    std::string filePath = "path/to/file.txt";

    if (fileExists(filePath)) {
        if (canAccessFile(filePath)) {
            // File exists and can be accessed
            // Proceed with accessing the file
            std::ifstream file(filePath);
            // Perform necessary operations on the file
            // ...
        }
    }
}
```

```

        } else {
            std::cout << "Cannot access the file." << std::endl;
        }
    } else {
        std::cout << "File does not exist." << std::endl;
    }

    return 0;
}

```

In the above code, the ***fileExists()*** function checks if the file exists by attempting to open it using an `std::ifstream` and checking if it is in a good state (`good()` function returns true). The ***canAccessFile()*** function checks if the file can be accessed by attempting to open it using an `std::ifstream` and checking if it is successfully opened (`is_open()` function returns true).

The existence of the file is checked using `fileExists`. If the file exists, it proceeds to check if it can be accessed using `canAccessFile`. By validating the file before accessing it, you can handle scenarios where the file doesn't exist or you don't have the required permissions to access it, preventing potential runtime errors. It is a good practice to use exception handling while opening a file. We can replace the if and else blocks by the try and catch blocks for the program to work efficiently.

2. Writing Files

When writing data into a file, the insertion operator (`<<`) and text wrapped in double-quotes are used with a `fstream` or `ofstream` object.

Example: Writing data to a file

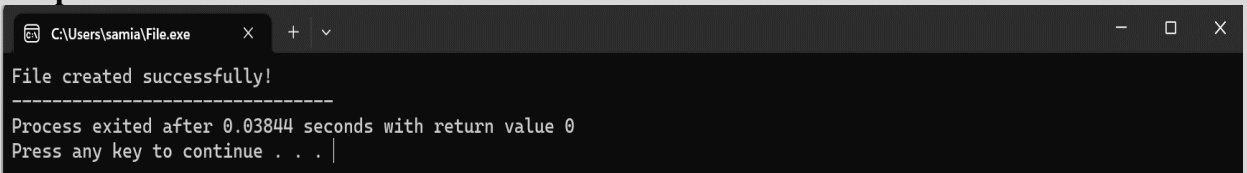
```

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

int main(){
    fstream my_file;
    my_file.open("file2", ios::out);
    if (!my_file){
        cout << "File not created!";
    }
    else{
        cout << "File created successfully!";
        my_file << "OOP BSAI - Section A";
        my_file.close();
    }
    return 0;
}

```

Output:

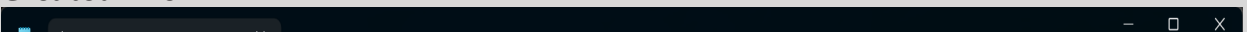


```

C:\Users\samia\file.exe
File created successfully!
-----
Process exited after 0.03844 seconds with return value 0
Press any key to continue . . .

```

Created File:



```

file2.txt
OOP BSAI - Section A

```

3. Reading Files

You can read information from files into your C++ program. This is possible using stream extraction operator (>>). You use the operator in the same way you use it to read user input from the keyboard. However, instead of using the cin object, you use the ifstream/ fstream object.

Example: Reading data from a file.

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

int main(){
    fstream my_file;
    my_file.open("file2.txt", ios::out);
    if (!my_file){
        cout << "No such file";
    }
    else {
        char ch;
        while(1){
            my_file >> ch;
            if (my_file.eof())
                break;
            cout << ch;
        }
    }
    my_file.close();

    return 0;
}
```

Here, after opening the file, we have taken a char variable and in a while loop starting from the beginning of the file till the end of file which is donated by an eof character. The *eof()* will generate true when eof character is reached which will terminate the loop otherwise the character on the file will be transferred to the output buffer.

Using Member Function getline

As its name states, read a whole line, or at least till a delimiter that can be specified

```
istream& getline(istream& is, string& str, char delim);
```

Parameters:

Mode	Description
is	It is an object of istream class and tells the function about the stream from where to read the input from.
str	It is a string object, the input is stored in this object after being read from the stream.
delim	It is the delimitation character which tells the function to stop reading further input after reaching this character.

```
istream& getline(istream& is, string& str);
```

The second declaration is almost the same as that of the first one. The only difference is, the latter has a delimitation character which is by default a new line(\n) character.

Example: Read a Line

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

int main(){
    fstream my_file;
    my_file.open("file2", ios::out);
    string line;
    if (!my_file){
        cout << "No such file";
    }
    else {
        string line;
        while(1){
            getline (my_file, line);
            cout << line;
        }
        my_file.close();
    }
    return 0;
}
```

Example: Read a character

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

int main(){
    ifstream openFile ("file.txt");
    char ch;
    if (!openFile){
        cout << "No such file";
    }
    else {
        while(!openFile.eof()){
            openFile.get(ch);
            cout << ch;
        }
    }
    openFile.close();
}
return 0;
}
```

Writing and Reading data in a file:

User provided data that we normally take input using cin can be written easily to a file and can also be read very easily. The example below writes data in a file and reads the data from file to display it on screen.

Example: Write user's Name and Age in a file


```

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

int main(){
    char data[100];

    //open a file in write mode
    ofstream outfile;
    outfile.open("array.dat");

    cout<< "Writing to the file" << endl;
    cout<< "Enter your name:";
    cin.getline(data, 100);

    //write input name into the file.
    outfile << data << endl;

    cout << "Enter your age:";
    cin >> data;
    cin.ignore();

    //again write input age into the file.
    outfile << data <<endl;

    //close the opened file
    outfile.close();

    //open file in read mode
    ifstream infile;
    infile.open ("array.dat");

    cout<<endl << "Reading data fom file" << endl;
    infile >> data;

    //write the name at screen
    cout << "Name:" << data << endl;

    //again read the age from the file and display it
    infile >> data;
    cout << "Age: " << data << endl;

    //close the opened file
    infile.close();

    return 0;
}

```

Examples make use of additional functions from cin object, like getline() function to read the line from outside and ignore() function to ignore the extra characters left by previous read statement

Output:

```

Writing to the file
Enter your name:Ali
Enter your age:20

```

```

Reading data from file

```

Write() function

The write() function is used to write object or record (sequence of bytes) to the file. A record may be an array, structure or class. Syntax of write() function is:

```
fstream fout;
fout.write((char *) &obj, sizeof(obj));
```

The write() function takes two arguments, **&obj** which refers to the initial byte of an object

stored in the memory and **sizeof(obj)** that provides the size of object represents the total number of bytes to be written from initial byte.

Example: Write () function

```
#include <iostream>
#include <fstream>
using namespace std;
class student{
    int roll;
    char name[25];
    float marks;
public:
    void getdata(){
        cout<< "Enter roll no:";
        cin >> roll;
        cout << "Enter name:" ;
        cin >> name;
        cout << "Enter marks:";
        cin >> marks;
    }

    void addRecord(){
        fstream f;
        student s;
        f.open("student.dat",ios::app | ios::binary );
        s.getdata();
        f.write((char*)&s, sizeof(s));
        f.close();
    }
};

int main(){
    student s;
    char c = 'n';
    do{
        s.addRecord();
        cout<<"Do you want to add another record.Press Y to continue:";
        cin >> c;
```

```

    } while (c == 'y' || c == 'Y');

    cout << endl << "Data written successfully";
}

```

Output:

```

Enter roll no:45
Enter name:Ali
Enter marks:67
Do you want to add another record. Press Y to continue: y

Enter roll no:98
Enter name:Jehadad
Enter marks:78
Do you want to add another record. Press Y to continue: n

Data written successfully

```

Read() function

The read() function is used to read object (sequence of bytes) to the file. Syntax of read() function is:

```

fstream fin;
fin.read((char *) &obj, sizeof(obj));

```

The read() function takes two arguments, **&obj** which refers to the initial byte of an object stored in the memory and **sizeof(obj)** that provides the size of object represents the total number of bytes to be written from initial byte. The read() function returns NULL if no data read.

Example: Read () function

```

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

class student{
    int roll;
    char name[25];
    float marks;
public:

    void displayStudent(){
        cout<<"Roll no: "<<roll <<endl;
        cout <<"Name: "<< name << endl;
        cout <<"Marks: "<<marks<<endl;
    }

    void Readdata()
    {
        fstream f;
        student s;
        f.open("studen.dat",ios::in | ios::binary);
        if(f.read((char*)&s,sizeof(s))){
            cout<<endl<<endl;
            s.displayStudent();
        }
    }
}

```

```
        else{
            cout<<"Error in reading data from file...\n";
        }
    }
};

int main(){
    student s;
    s.Readdata();
}
```

Output:

```
Enter roll no:45
Enter name:Ali
Enter marks:67
```

Exercise

1. Write a program to implement I/O operations on characters. I/O operations includes inputting a string, calculating length of the string, Storing the String in a file and fetch the stored characters from it.
2. Write a program to copy the contents of one file to another.
3. Take a class Person having two attributes name and age. Include a parametrized constructor to give values to all data members. In main function Create an instance of the person class and name it person1. Create a binary file person.bin and write person1 object into it. Read the person1 object from the file.
4. Take a class Participant having three attributes (ID, name and score) and following member functions.
 - ✓ Input () function takes data of the object and stores it in a file name participant.dat
 - ✓ Output () function takes id from user and show respective data of that id.
 - ✓ Max () gives the highest score of the Participant in the file.
5. Write a function in C++ to count and display the number of lines not starting with alphabet 'A' present in a text file "STORY.TXT".
Example: If the file "STORY.TXT" contains the following lines,

```
The rose is red.  
A girl is playing there.  
There is a playground.  
An airplane is in the sky.  
Numbers are not allowed in the password.
```

The function should display the output as 3.