Files
Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it. A stream is an abstraction that represents a device on which operations of input and output are performed.

Types of files
A text file is the one in which data is stored in the form of ASCII characters and is normally used for storing a stream of characters. Text files are organized around lines, each of which ends with a newline character ('\n'). The source code files are themselves text files.

A binary file is the one in which data is stored in the file in the same way as it is stored in the main memory for processing. It is stored in binary format instead of ASCII characters. It is normally used for storing numeric information (int, float, double). Normally a binary file can be created only from within a program and its contents can be read only by a program.

What is 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.

The term "Data" is commonly referred to as known facts or information. In the present era, data plays a vital role. It helps to describe, diagnose, predict or prescribe. But to achieve all this, we need to store it somewhere. You all would argue that there are so many text editors like 'Notepad' and 'MS Office', which help us store data in the form of text. You are right! But here we are discussing at a level of programming. In contrast, text editors like 'Notepad' and 'MS Office' are pre-built and cannot be accessed at the programming level to store data. File Handling is a hot topic when it comes to storing such programming data.

Almost every programming language has a 'File Handling' method to deal with the storage of data. In this article, we will learn about file handling in C++. But, before that, if you are a newbie at C++, you could check out this free course on C++ to learn the basics.
Now, This topic of file handling is further divided into sub-topics:

1. Create a file
2. Open a file
3. Read from a file
4. Write to a file
5. Close a file

fstream library
Before diving into each sub-topics, let us first learn about the header file we will be using to gain access to the file handling method. In C++, fstream library is used to handle files, and it is dealt with the help of three classes known as ofstream, ifstream and fstream.

ofstream: This class helps create and write the data to the file obtained from the program's output. It is also known as the input stream.

Ifstream: We use this class to read data from files and also known as the input stream.

fstream: This class is the combination of both ofstream and ifstream. It provides the capability of creating, writing and reading a file.

To access the following classes, you must include the fstream as a header file like how we declare iostream in the header.

Example:

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

After including the header file, there comes a question saying do we need to create the file within the program or else do we need to use an existing file. But this isn't that difficult to answer because, in C++, we get four different methods to handle files. Let's discuss them one by one.

File Operations in C++
C++ provides us with four different operations for file handling. They are:
open() – This is used to create a file.
read() – This is used to read the data from the file.
write() – This is used to write new data to file.
close() – This is used to close the file.
We will look into each of these and try to understand them better.

Opening files in C++
To read or enter data to a file, we need to open it first. This can be performed with the help of 'ifstream' for reading and 'fstream' or 'ofstream' for writing or appending to the file. All these three objects have open() function pre-built in them.

Syntax:

```
open( FileName , Mode );
```

FileName – It denotes the name of a file which has to be opened.

Mode – There are different modes to open a file

Mode	**Description**
ios::in	File opened in reading mode

ios::out	File opened in write mode
ios::app	File opened in append mode
ios::ate	File opened in append 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

Program for Opening File:

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
   fstream FileName;
   FileName.open("FileName", ios::out);
   if (!FileName){
      cout<<"Error while creating the file";
   }
   else{
      cout<<"File created successfully";
      FileName.close();
   }
   return 0;
}
```

Explanation of above code

1. Here we have an iostream library, which is responsible for input/output stream.
2. We also have a fstream library, which is responsible for handling files.
3. Creating an object of the fstream class and named it as 'FileName'.
4. On the above-created object, we have to apply the open() function to create a new file, and the mode is set to 'out' which will allow us to write into the file.
5. We use the 'if' statement to check for the file creation.
6. Prints the message to console if the file doesn't exist.
7. Prints the message to console if the file exists/created.
8. We use the close() function on the object to close the file.

Output

```
File created successfully
```

Writing to File

Till now, we learned how to create the file using C++. Now, we will learn how to write data to file which we created before. We will use fstream or ofstream object to write data into the file and to do so; we will use stream insertion operator (<<) along with the text enclosed within the double-quotes.

With the help of open() function, we will create a new file named ‘FileName’ and then we will set the mode to ‘ios::out’ as we have to write the data to file.

Syntax:

```
FileName<<"Insert the text here";
```

Program for Writing to File:

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
   fstream FileName;
   FileName.open("FileName.txt", ios::out);
   if (!FileName) {
      cout<<" Error while creating the file ";
   }
   else {
      cout<<"File created and data got written to file";
      FileName<<"This is a blog posted on <a href="https://www.mygreatlearning.com"
data-internallinksmanager029f6b8e52c="11" title="Great Learning Homepage" target="_blank"
rel="noopener">Great Learning</a>";
      FileName.close();
   }
   return 0;
}
```

Explanation of above code

1. Here we have an iostream library, which is responsible for input/output stream.
2. We also have a fstream library, which is responsible for handling files.
3. Creating an object of the fstream class and named it as ‘FileName’.
4. On the above-created object, we have to apply the open() function to create a new file, and the mode is set to ‘out’ which will allow us to write into the file.
5. We use the ‘if’ statement to check for the file creation.
6. Prints the message to console if the file doesn’t exist.

7. Prints the message to console if the file exists/created.
8. Writing the data to the created file.
9. We use the close() function on the object to close the file.

Output
File created and data got written to file

Reading from file in C++
Getting the data from the file is an essential thing to perform because without getting the data, we cannot perform any task. But don't worry, C++ provides that option too. We can perform the reading of data from a file with the CIN to get data from the user, but then we use CIN to take inputs from the user's standard console. Here we will use fstream or ifstream.

Syntax:
FileName>>Variable;
Content of FileName.txt:
Hello World, Thank You for Creating Me!.

Program for Reading from File:
```
#include<iostream>
#include <fstream>
using namespace std;
int main() {
   fstream FileName;
   FileName.open("FileName.txt", ios::in);
   if (!FileName) {
      cout<<"File doesn't exist.";
   }
   else {
      char x;
      while (1) {
         FileName>>x;
         if(FileName.eof())
            break;
         cout<<x;
      }
   }
   FileName.close();
   return 0;
}
```

Explanation of above code
1. Here we have an iostream library, which is responsible for input/output stream.
2. We also have a fstream library which is responsible for handling files.

3. Creating an object of the fstream class and named it ‘FileName’.
4. On the above-created object, we have to apply the open() function to create a new file, and the mode is set to ‘in’ which will allow us to read from the file.
5. We use the ‘if’ statement to check for the file creation.
6. Prints the message to console if the file doesn’t exist.
7. Creating a character(char) variable with the named x.
8. Iterating of the file with the help of while loop.
9. Getting the file data to the variable x.
10. Here we are using if condition with eof() which stands for the end of the file to tell the compiler to read till the file’s end.
11. We use the ‘break’ statement to stop the reading from file when it reaches the end.
12. The print statement to print the content that is available in the variable x.
13. We use the close() function on the object to close the file

Output

```
Hello World, Thank You for Creating Me!
```

Closing a file in C++

Closing a file is a good practice, and it is must to close the file. Whenever the C++ program comes to an end, it clears the allocated memory, and it closes the file. We can perform the task with the help of close() function.

Syntax:

```
FileName.close();
```

Program to Close a File:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   fstream FileName;
   FileName.open("FileName.txt", ios::in);
   if (!FileName) {
      cout<<"File doesn’t exist";
   }
   else {
      cout<<"File opened successfully";
      }
   }
   FileName.close();
   return 0;
}
```

Explanation of above code

1. Here we have an iostream library, which is responsible for input/output stream.
2. We also have a fstream library, which is responsible for handling files.
3. Creating an object of the fstream class and named it as ‘FileName’.
4. On the above-created object, we will apply the open() function to create a new file, and the mode is set to ‘out’ which allows us to write into the file.
5. We use the ‘if’ statement to check for the file creation.
6. Prints the message to console if the file doesn’t exist.
7. Prints the message to console if the file opened or not.
8. We use the close() function on the object to close the file.

1. Files:

- Definition: A file is a collection of data or information stored on a storage device, such as a hard drive or SSD, under a specific name and location.
- Types of Files:
  - Text Files: Text files store data in the form of plain text, consisting of characters encoded using a character encoding scheme such as ASCII or UTF-8. Examples include .txt files.
  - Binary Files: Binary files store data in a format that is not human-readable, consisting of sequences of binary digits (0s and 1s). They can store any type of data, including images, audio, video, and executable programs.

2. Stream Classes and Their Member Functions:

- ifstream (Input File Stream):
  - Member Functions:
    - get(): Reads a single character from the input stream.
    - getline(): Reads a line of text from the input stream.
    - read(): Reads a specified number of bytes from the input stream.
    - seekg(): Sets the position of the file pointer in the input stream.
    - tellg(): Returns the current position of the file pointer in the input stream.
    - open(): Opens a file for input.
    - close(): Closes the input file.
    - eof(): Returns true if the end of the file has been reached.
- ofstream (Output File Stream):
  - Member Functions:
    - put(): Writes a single character to the output stream.
    - seekp(): Sets the position of the file pointer in the output stream.
    - tellp(): Returns the current position of the file pointer in the output stream.
    - write(): Writes a specified number of bytes to the output stream.
    - open(): Opens a file for output.
    - close(): Closes the output file.
- fstream (File Stream):

    - Inherits Member Functions from ifstream and ofstream: fstream inherits all member functions from both ifstream and ofstream, allowing it to handle both input and output operations on files.

3. File Modes (ios flags):

- ios::app: Append mode. Data is written to the end of the file.
- ios::ate: Set the file pointer to the end of the file when opening.
- ios::in: Open file for reading.
- ios::out: Open file for writing.
- ios::binary: Open file in binary mode.
- ios::trunc: Truncate the file if it already exists.
- ios::nocreate: Do not create the file if it does not exist.
- ios::noreplace: Do not overwrite the file if it already exists.

4. Opening a File & Closing a File

- Files can be opened using either the constructor or the open() member function.
- Files should be closed using the close() member function to release system resources and ensure data integrity.

```
#include <iostream>
#include <fstream> // Required for file handling
using namespace std; // Add the directive to use the std namespace
int main() {
  // Using constructor to open a file
  ofstream file1("example.txt"); // Opens "example.txt" for writing

  // Check if the file is open
  if (file1.is_open()) {
    cout << "File opened successfully using constructor.\n";
    // File operations here
    file1 << "Hello, World!\n"; // Write to file
```

```
        file1.close(); // Close the file
    } else {
        cout << "Error opening file using constructor!\n";
        return 1; // Exit with error code
    }

    // Using open() member function to open a file
    ofstream file2; // Declare file stream object
    file2.open("example2.txt"); // Opens "example2.txt" for writing

    // Check if the file is open
    if (file2.is_open()) {
        cout << "File opened successfully using open() member function.\n";
        // File operations here
        file2 << "Hello again, World!\n"; // Write to file
        file2.close(); // Close the file
    } else {
        cout << "Error opening file using open() member function!\n";
        return 1; // Exit with error code
    }
    return 0; // Exit with success
}
```

6. Detecting the End of a File:

- The eof() member function can be used to detect the end of a file.

```
#include <iostream>

#include <fstream> // Required for file handling

using namespace std;

int main() {

   ifstream file("example.txt"); // Open "example.txt" for reading

   // Check if the file is open

   if (!file.is_open()) {

      cout << "Error opening file!\n";

      return 1; // Exit with error code if file cannot be opened

   }

   char ch;

   // Read characters until end of file is reached

   while (!file.eof()) {

      file.get(ch); // Read a character from the file

      // Check if character is not end-of-file marker

      if (!file.eof()) {

         cout << ch; // Output character

      }

   }
```

```
    file.close(); // Close the file

    return 0; // Exit with success

}
```

7. File Pointer and Their Manipulation:

- The file pointer indicates the current position in the file. It can be manipulated using seekg() and seekp().

```
#include <iostream>

#include <fstream> // Required for file handling

using namespace std;

int main() {

    // Writing data to a file

    ofstream outFile("example.txt");

    if (!outFile.is_open()) {

        cout << "Error opening file for writing!\n";

        return 1;

    }

    outFile << "This is line 1.\n";

    outFile << "This is line 2.\n";

    outFile << "This is line 3.\n";

    outFile.close();

    // Reading data from the file and manipulating file pointer

    ifstream inFile("example.txt");
    if (!inFile.is_open()) {
        cout << "Error opening file for reading!\n";
        return 1;
    }
    // Set the file pointer to the beginning of the third line
    inFile.seekg(0, ios::beg); // Move to the beginning of the file
    inFile.seekg(0, ios::beg); // Move to the beginning of the file (redundant)
    inFile.seekg(0, ios::cur); // Move 0 characters from the current position (redundant)
    inFile.seekg(0, ios::end); // Move to the end of the file (redundant)
    inFile.seekg(0, ios::end); // Move to the end of the file (redundant)
    inFile.seekg(0, ios::cur); // Move 0 characters from the current position (redundant)
    inFile.seekg(0, ios::beg); // Move to the beginning of the file (redundant)
    // Read and display the third line
    string line;
    getline(inFile, line); // Read the first line
    getline(inFile, line); // Read the second line
    cout << "Third line: " << line << endl; // Output the third line
    inFile.close();
    return 0;
}
```

8. Text Files:

- Creation: Text files can be created using ofstream or fstream and written to using << operator or write() member function.
- Display: Text files can be displayed using ifstream and read from using >> operator or getline() member function.
- File Processing: Character and string-based processing can be performed using ifstream and ofstream member functions.

```
#include <iostream>

#include <fstream> // Required for file handling

using namespace std;

int main() {

   // Creating a text file and writing data to it

   ofstream outFile("example.txt"); // Open "example.txt" for writing

   if (!outFile.is_open()) {

      cout << "Error opening file for writing!\n";

      return 1; // Exit with error code if file cannot be opened

   }

   outFile << "This is line 1.\n";

   outFile << "This is line 2.\n";

   outFile << "This is line 3.\n";

   outFile.close(); // Close the file

   // Displaying the contents of the text file

   ifstream inFile("example.txt"); // Open "example.txt" for reading

   if (!inFile.is_open()) {

      cout << "Error opening file for reading!\n";

      return 1; // Exit with error code if file cannot be opened
   }

   cout << "Contents of the file:\n";

   char ch;

   while (inFile.get(ch)) { // Read characters from file until end-of-file

      cout << ch; // Output each character

   }

   inFile.close(); // Close the file

   // Processing the text file (Character-based)

   ifstream processFile("example.txt"); // Open "example.txt" for reading

   if (!processFile.is_open()) {

      cout << "Error opening file for reading!\n";

      return 1; // Exit with error code if file cannot be opened

   }

   cout << "\n\nProcessing the file (Character-based):\n";

   int characterCount = 0;

   while (processFile.get(ch)) { // Read characters from file until end-of-file

      characterCount++; // Increment character count for each character read

   }

   cout << "Number of characters in the file: " << characterCount << endl;

   processFile.close(); // Close the file

   // Processing the text file (String-based)

   ifstream processStringFile("example.txt"); // Open "example.txt" for reading
```

```
    if (!processStringFile.is_open()) {

        cout << "Error opening file for reading!\n";

        return 1; // Exit with error code if file cannot be opened

    }

    cout << "\nProcessing the file (String-based):\n";

    string line;

    int lineCount = 0;

    while (getline(processStringFile, line)) { // Read lines from file until end-of-file

        lineCount++; // Increment line count for each line read

        cout << "Line " << lineCount << ": " << line << endl; // Output each line

    }

    cout << "Number of lines in the file: " << lineCount << endl;

    processStringFile.close(); // Close the file

    return 0; // Exit with success

}
```

9. Binary Files:

- Creation: Binary files can be created using ofstream or fstream and written to using write() member function.
- Display: Binary files can be displayed using ifstream and read from using read() member function.
- File Processing: Appending, inserting, deleting, updating, searching, splitting, and merging operations can be performed on binary files.

```
#include <iostream>
#include <fstream> // Required for file handling
#include <iomanip> // Required for formatting output
```

```
using namespace std;
// Structure to represent a record in the binary file
struct Record {
    int id;
    char name[50];
    float salary;
};

// Function to display a record
void displayRecord(const Record& rec) {
    cout << "ID: " << rec.id << ", Name: " << rec.name << ", Salary: $" << fixed << setprecision(2)
<< rec.salary << endl;
}

// Function to create and write records to a binary file
void createBinaryFile(const char* filename) {
    // Create an array of records
    Record records[] = {
        {1, "John Doe", 50000.0f},
        {2, "Jane Smith", 60000.0f},
        {3, "Alice Johnson", 70000.0f}
    };

    // Open the binary file for writing
    ofstream outFile(filename, ios::binary);
    if (!outFile.is_open()) {
        cout << "Error opening file for writing!\n";
        return;
    }

    // Write records to the file
    for (const auto& rec : records) {
        outFile.write(reinterpret_cast<const char*>(&rec), sizeof(Record));
    }

    // Close the file
    outFile.close();
}

// Function to display all records in a binary file
void displayBinaryFile(const char* filename) {
    // Open the binary file for reading
    ifstream inFile(filename, ios::binary);
    if (!inFile.is_open()) {
        cout << "Error opening file for reading!\n";
        return;
    }

    Record rec;

    // Read and display all records from the file
    while (inFile.read(reinterpret_cast<char*>(&rec), sizeof(Record))) {
        displayRecord(rec);
    }

    // Close the file
    inFile.close();
}

int main() {
    const char* filename = "example.bin";

    // Create and write records to the binary file
    createBinaryFile(filename);

    // Display all records in the binary file
    cout << "Records in the binary file:\n";
    displayBinaryFile(filename);

    return 0;
}
```