C++ File Handling

File handling in C++ is a mechanism to create and perform read/write operations on a file.

We can access various file handling methods in C++ by importing the <fstream> class.

```
#include <fstream>
```

<fstream> includes two classes for file handling:

- ifstream - to read from a file.
- ofstream - to create/open and write to a file.

Note: Our online compiler cannot handle file handling right now. So, please install an IDE or text editor on your computer to run the programs given here.

Opening and Closing a File

In order to work with files, we first need to open them. In C++, we can open a file using the ofstream and ifstream classes.

For example, here's how we can open a file using ofstream:

```
std::ofstream my_file("example.txt");
```

Here,

- my_file - the name of the object of the ofstream class.
- example.txt - the name and extension of the file we want to open.

Note: We can also use the open() function to open a file. For example,

```
std::ofstream my_file.open("example.txt");
```

Closing a File

Once we're done working with a file, we need to close it using the close() function.

```
my_file.close();
```

Let's take an example program to look at these operations.

Example 1: Opening and Closing a File

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

using namespace std;

int main() {

   // opening a text file for writing
   ofstream my_file("example.txt");

   // close the file
   my_file.close();

   return 0;
}
```

This code will open and close the file example.txt.

> Note: If there's no such file to open, ofstream my_file("example.txt"); will instead create a new file named example.txt.

Check the File for Errors

In file handling, it's important to ensure the file was opened without any error before we can perform any further operations on it.There are three common ways to check files for errors:

1. By Checking the File Object

```
ofstream my_file("example.txt");

// check if the file has been opened properly
```

```
if (!my_file) {

   // print error message
   cout << "Error opening the file." << endl;

   // terminate the main() function
   return 1;
}
```

Notice the condition in the if statement:

```
if (!my_file) {...}
```

This method checks if the file is in an error state by evaluating the file object itself.

- If the file has been opened successfully, the condition evaluates to true.
- If there's an error, it evaluates to false, and you can handle the error accordingly.

2. Using the is_open() Function

The is_open() function returns

- true - if the file was opened successfully.
- false - if the file failed to open or if it is in a state of error.

For example,

```
ofstream my_file("example.txt");

if (!my_file.is_open()) {
   cout << "Error opening the file." << endl;
   return 1;
}
```

3. Using the fail() Function

The fail() function returns

- true - if the file failed to open or if it is in a state of error.
- false - if the file was opened successfully.

```
ofstream my_file("example.txt");

if (my_file.fail()) {
    cout << "Error opening the file." << endl;
    return 1;
}
```

Note: For simplicity, we recommend using the first method.

Read From a File

Reading from text files is done by opening the file using the ifstream class. For example,

```
ifstream my_file("example.txt");
```

Then, we need to read the file line-by-line. To do this, we need to loop through each line of the file until all the lines are read, i.e., until we reach the end of the file.

We use the eof() function for this purpose, which returns

- true - if the file pointer points to the end of the file
- false - if the file pointer doesn't point to the end of the file

For example,

```
// variable to store file content
string line;

// loop until the end of the file
while (!my_file.eof()) {

    // store the current line of the file
    // in the "line" variable
    getline(my_file, line);

    // print the line variable
    cout << line << endl;
}
```

Here, the while loop will run until the end of the file. In each iteration of the loop,

- getline(my_file, line); reads the current line of the file and stores it in the line variable.
- Then, it prints the line variable.

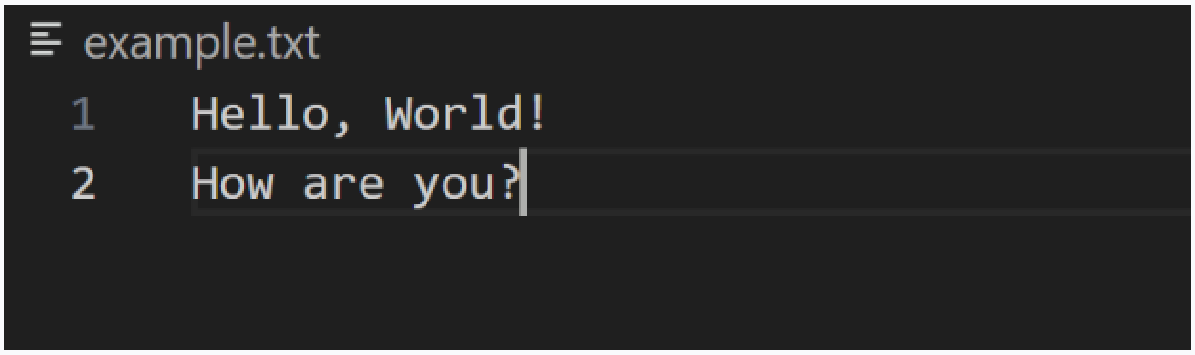
Next, let's clarify this with a working example.

Example 2: Read From a File

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

using namespace std;

int main() {

   // open a text file for reading
   ifstream my_file("example.txt");

   // check the file for errors
   if(!my_file) {
      cout << "Error: Unable to open the file." << endl;
      return 1;
   }

   // store the contents of the file in "line" string
   string line;

   // loop until the end of the text file
   while (!my_file.eof()) {

      // store the current line of the file
      // in the "line" variable
      getline(my_file, line);

      // print the line variable
      cout << line << endl;
```

```
    }

    // close the file
    my_file.close();

    return 0;
}
```

Suppose example.txt contains the following text:


example.txt
```
Hello, World!
How are you?
```

Contents of example.txt
Then, our terminal will print the following output:

```
Hello, World!
How are you?
```

Write to a File

We use the ofstream class to write to a file. For example,

```
ofstream my_file("example.txt");
```

We can then write to the file by using the insertion operator << with the ofstream object my_file. For example,

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

using namespace std;

int main() {
    // open a text file for writing
    ofstream my_file("example.txt");

    // check the file for errors
    if(!my_file) {
        cout << "Error: Unable to open the file." << endl;
        return 1;
    }

    // write multiple lines to the file
    my_file << "Line 1" << endl;
    my_file << "Line 2" << endl;
    my_file << "Line 3" << endl;

    // close the file
    my_file.close();

    return 0;
}
```

Notice the following code for writing to the file:

```
my_file << "Line 1" << endl;
my_file << "Line 2" << endl;
my_file << "Line 3" << endl;
```

This is similar to printing output to a screen:

```
cout << "Line1" << endl;
```

In file handling, we just replace cout with the file object to write to the file instead of the console.

Our particular code will write the following text to example.txt:

```
Line1
Line2
Line3
```

Note: Writing to an existing file will overwrite the existing contents of the file.

Append to a Text File

To add/append to the existing content of a file, you need to open the file in append mode.

In C++, you can achieve this by using the ios::app flag when opening the file:

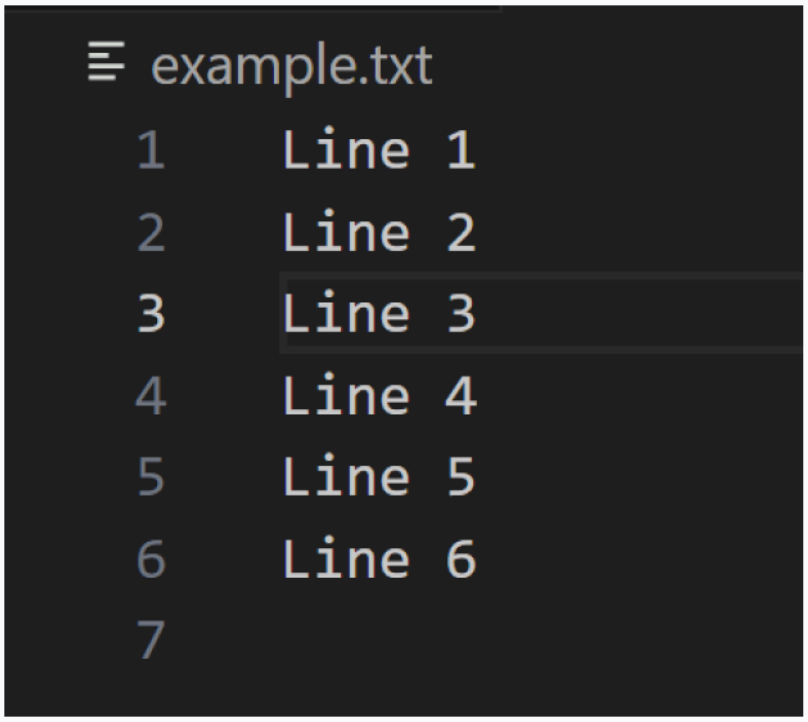
```
ofstream my_file("example.txt", ios::app);
```

Now, let's add some more text to the existing content of example.txt:

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

using namespace std;

int main() {

    // open a text file for appending
    ofstream my_file("example.txt", ios::app);

    // if the file doesn't open successfully, print an error message
    if(!my_file) {
        cout << "Failed to open the file for appending." << endl;
        return 1;
    }

    // append multiple lines to the file
    my_file << "Line 4" << endl;
    my_file << "Line 5" << endl;
    my_file << "Line 6" << endl;

    // close the file
    my_file.close();

    return 0;
}
```

This will add the following lines to example.txt:

```
Line 4
Line 5
Line 6
```



```
example.txt
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6

```

Contents Appended to an Existing File

File Handling With fstream

Instead of using ifstream to read from a file and ofstream to write to the file, we can simply use the fstream class for all file operations.

The constructor for fstream allows you to specify the file name and the mode for file operations.

Mode	Description

ios::in	Opens the file to read (default for ifstream).
ios::out	Opens the file to write (default for ofstream).
ios::app	Opens the file and appends new content to itat the end.

Let's look at an example:

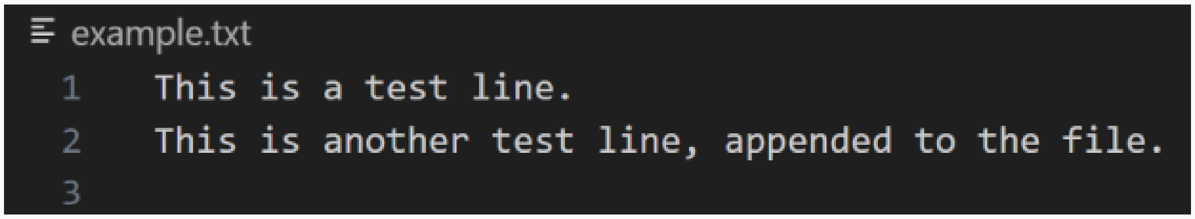
```
#include <iostream>
#include <fstream>

using namespace std;

int main() {

   // 1. write to a text file
   fstream my_file("example.txt", ios::out);

   if (my_file) {
      my_file << "This is a test line." << endl;
      my_file.close();
   }
   else {
      cout << "Unable to open file for writing." << endl;
      return 1;
   }

   // 2. read from the same file
   string line;
   my_file.open("example.txt", ios::in);

   if (my_file) {
      while (!my_file.eof()) {
         getline(my_file, line);
         cout << "Read from file: " << line << endl;
```

```
        }
        my_file.close();
    }
    else {
        cout << "Unable to open file for reading." << endl;
        return 1;
    }

    // 3. append data to the end of the file
    my_file.open("example.txt", ios::app);

    if (my_file) {
        my_file << "This is another test line, appended to the file." << endl;
        my_file.close();
    }
    else {
        cout << "Unable to open file for appending." << endl;
        return 1;
    }

    return 0;
}
```

Output

```
Read from file: This is a test line.
Read from file:
```

If we look at the file after running the program, we will find the following contents:



```
example.txt
This is a test line.
This is another test line, appended to the file.
```