**Chapter 14: File Handling**

In this chapter we will be taking a look at sequential file handling of text files. It is important to note that any data you set when running your programs will be lost once the program exits. There are many times that you need to retain information between runs of your program. An example would be a high score table. If the high scores were lost each time the application shut down it wouldn’t be very interesting for the player. This information will be stored in a file and retrieved upon loading.

To be able to use any of the file handling functionality built into C++ you must include fstream and iostream at the top of your code files. Iostream is required as fstream inherits from this and uses the std namespace.



All the below examples have these included.

When opening a file in C++ you must state which operations you wish to perform on the file. This is determined by the iosMode that is set in the constructor. We will be looking at the constructor shortly, but first we need to understand what iosModes are available to us. Look at the table below for the iosMode Values.

Default values can be used when using the various fstreams available, but more often than not you will wish to set the iosMode yourself. An example would be if you needed to append data at the end of a file instead of overwriting the data within.

|  |  |
| --- | --- |
| **iosMode** | **Description** |
| ios::in | Open file for input. |
| ios:out | Open file for output. |
| ios::binary | Open file in binary mode. |
| ios::ate | Set the initial position of the file pointer to the end of the file. |
| ios::app | All operations are done at the end of the file, appending any data to the current contents of the file. |
| ios::trunc | If the file is opened for output and it already exists, the previous version is deleted and replaced with this new one. |

At times you will need to set multiple iosModes. To do this you use the ‘|’ symbol (a single bar as that used in an OR condition). An example of opening a file for output and to overwrite an existing file would look like this:



**Opening a File**

To open a file, you first need to determine whether the file is going to be read from or written to. Depending on this choice there are three data type options. See table below:

|  |  |
| --- | --- |
| Class | Default iosMode |
| ifstream | ios::in |
| ofstream | ios::out |
| fstream | ios::in | ios::out |

As can be seen from the table, default values are set depending on the data type. These can be changed in the constructor.

To open a file for input use one of the following approaches:

Using the default:



Overriding the iosMode:



To open a file for output use one of the following approaches:

Using the default:



Overriding the iosMode:



To open a file for input and output use on of the following approach which uses the default iosMode:



You can also use the open() function, which accepts the name of the file you want to open along with the iosMode flags.



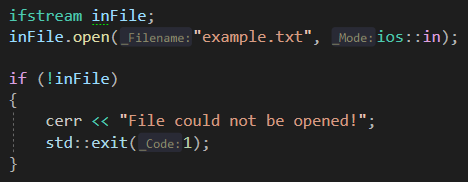
**Closing a file**

To close a file it is simply a matter of calling the close() function:

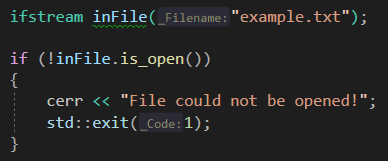


**Checking a file was opened**

Before attempting to read or write to a file it is important to ensure that the file was successfully opened. To do this use one of the following approaches:



Alternatively use the is is\_open() function:



Note: exit(1) will end the program. This works is tangent with return 0; Returning 0 says that’s the program was a success and exit ones fails the program.

**Writing to a file**

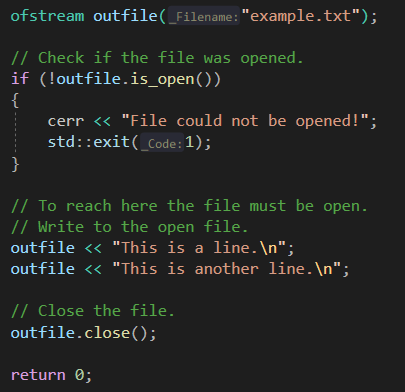
To write to a file you simply use the >> operator in the same way as you have done in previous example with cout.



**Example of creating a file and saving data**

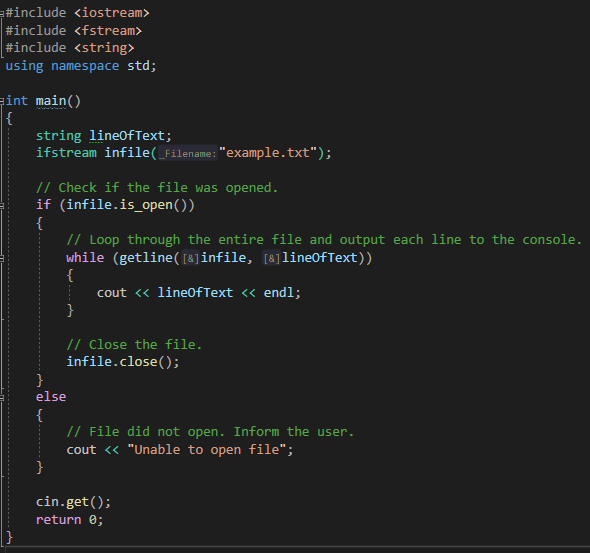
It is recommended you replicate these examples to help understand what is happening.

When you recreate this program, it may seem as though nothing happens, it will end the program rather quickly. However, once run check your project folder where the cpp file and you will find a .txt file has been created, open it and ensure the statements you passed to the file are there.



**Reading from a file**

When working with this you need to ensure that you copy the file you created in the last program and paste it in this projects folder as we are not creating a file in this program but rather reading an existing one.



**Moving the file Pointer**

When first opening a file the file pointer will be positioned at the beginning of the file. At times you will need to move this pointer. Functions have been provided which enable this. seekg() is used with ifstream and seekp() for ofstream. You need to pass into the function the number of bytes you wish to offset from the beginning of the file.

To seek n bytes from the beginning of the file:



To seek n bytes from the current position of the file:



To seek back n bytes from the end of the file:

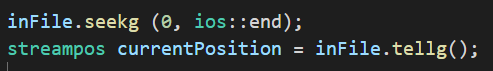


To seek to the end of the file:



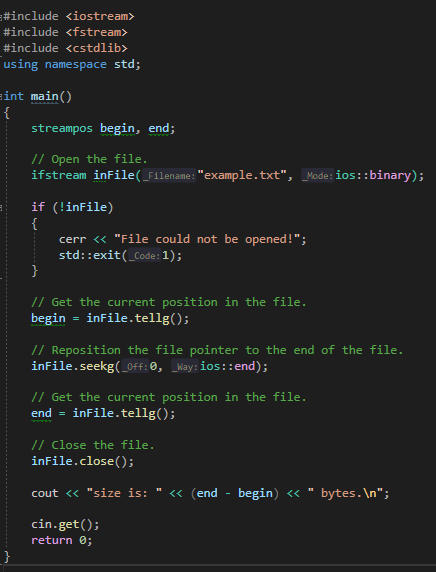
When using the tellg() or tellp() functions you will notice that they return a new data type. This is the streampos type. This type can hold the internal position of the file pointer.

To seek to the end of a file and return the streampos:



**Example of file size**

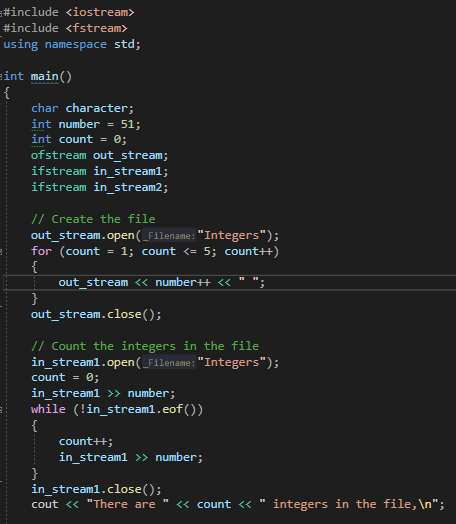
This code listing will demonstrate how you can use the tellg() function to determine the size of the file. Copy across the example.txt file from the previous program. It needs to be placed in the project folder.

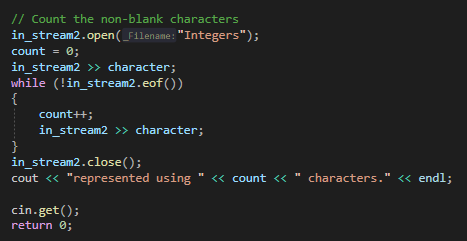


Run the program and make sure it works. It should output “size is: 40 bytes”.

**Example of counting characters**

This code listing will read and write files using the different streams available. It will count the number of integers in the file and the number of non-blank characters.





Replicate and run the program and ensure you get the following output:



**Program 38: Account Holding**

Reading the instructions and the pseudocode carefully. This program will take what you learnt last week and incorporate file handling. You will be creating a simple database that holds some account details: name, account number and balance. This program will give the user four choices:

* To enter new details
* Display the contents of the file
* Display accounts that have a balance over 10,000
* Exit the program

You will be expected to create a struct to hold data and use two instances of that struct, one to hold the input from the user and one to hold the data from reading in from the file and output the results from. The file is expected to append the file.

Some basic pseudocode is supplied below to help get you started, there are other ways to do it.

It is expected you will include the appropriate libraries.

Add struct above the main, this should contain:

string name

int account number

double balance

in the main:

create two instances of the struct

create two bools for the while loops

create an int for the user’s choice and a char for if the user wants to enter more details.

Create an outFile and inFile using ofstream and ifstream

Create a while loop

In the loop, output the options and store their choice.

You can use a switch or else-if this example uses an else if

If user choice equals 1

Create/open a file this will require 2 modes. The file **must** be **appended**

While loop

If file is open

Ask for details and store in first struct instance e.g. p.name, p.acc\_num, p.balance

Push answers to the file e.g. outFile << p.name << endl;

Ask if they wish to enter more details

If no end loop

Close file

If choice equals 2

Reset the loop bool if they chose not to enter more details so they can again

open file

if file open

while loop save the details the second struct instance created

output details from the second struct instance

close file

if choice equals 3

repeat if choice is two up to the while loop

if balance > 100000

output details

else

inform that no accounts exist with a balance over 10000

Note: the while loop in choices 2 and 3, the condition may trip some up. You may try **!eof()** as a condition which checks for the end of the file, however, this can cause issues like repeating the last entry. This will give an incorrect output. So, your condition should check while inFile is filling struct data. E.g inFile >> savedAccounts.name, etc.

Else If choice equals 4

exit program

Else

choice anything but 1,2,3 or 4. Inform of unknown selection

repeat loop giving choices to the user

**Program 38 Source code:**

**Program 38 Screenshot of output**

Screenshots must show file is created, output all contents (option 2) and accounts that are greater than 10000 (option 3)

**Program 39: Top Ten Scores**

This program is to test not only your file handling ability but also how you choose to handle the data internally. Only 10 scores and names should ever be stored. A screenshot of the scores text file should also be included below. It **MUST** match the output expected from the screenshot of the program running.

Write a program that presents the user with the following option screen:

1. Enter a score

2. Display scores

3. Exit

If the user enters an invalid option, the program should inform the user and then re-present the menu screen.

Upon entering ‘1’ the user should be asked for a score and a name. If the entered score is greater than the lowest score already in the file, or there are less than 10 scores stored this new score should be incorporated in to the top ten. This should then be saved in a text file called ‘scores.txt’. The user is then returned to the menus screen.

Upon entering ‘2’ the program should output a **well-formatted** list of scores, use **iomanip** library to help with this. This should stay on screen until the user presses the return key. If there are no scores the user should be informed and then returned to the menu screen.

Upon entering ‘3’ the program should close. All file streams should be closed correctly.

NOTE: This program must not crash. It is up to you as programmer to ensure invalid options are dealt with and that if the file doesn’t exist and the user tries to access it that they are informed, and the program returns to the menu. The use of functions and structs will be useful. Possible functions: enterScore, displayScore, countScores, savedScores and loadScores.

**Program 39 Source code**

**Program 39 Output Screenshots**

Screenshots should show multiple entries