Chapter 3

File

File and stream

- 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.
- In this chapter we will learn how to read and write from a file.
- cin is an object of class istream and cout is an object of class ostream.
- we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files.
- This requires another standard C++ library called **fstream**, which defines three new data types –

Sr.No	Data Type & Description
1	**ofstream** This data type represents the output file stream and is used to create files and to write information to files.
2	**ifstream** This data type represents the input file stream and is used to read information from files.
3	**fstream** This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

Create and Write To a File

- To use the fstream library, include both the standard <iostream> **AND** the <fstream> header file.
- To create a file, use either the ofstream or fstream class, and specify the name of the file.
- Steps:-

1. Include the necessary headers: iostream for input/output and fstream for file I/O.
2. Create an output file stream object using the ofstream class.
3. Open the file for writing using the open() method of the output file stream object.
4. Write data to the file using the << operator, just like writing to cout.
5. Close the file using the close() method of the output file stream object.

Step 1:Include the necessary headers: iostream for input/output and fstream for file I/O.

Example:

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

Step2: Create an output file stream object using the ofstream class.

Syntax:

```
ofstream objectname;
```

Example: ofstream myfile;

- object of the ofstream class called myfile, has declared but doesn't specify a filename yet.

- Step3:- Open the file for writing using the open() method of the output file stream object.

Syntax : Objectname open(filename.txt);

Example:

```
ofstream outputFile;
outputFile.open("filename.txt");
```

- In this example, we declare an object of the ofstream class called outputFile.
- We then call the open() method on the outputFile object and pass in the name of the file we want to open for writing ("filename.txt" in this case).
- If the file doesn't exist, it will be created. If the file already exists, it will be truncated (i.e., its contents will be deleted) and then opened for writing.

Step 4:Write data to the file using the << operator, just like writing to cout.

Syntax: object_name<<something to be written to file;

Example:

```
ofstream outputFile;
 outputFile.open("filename.txt");
 outputFile << "This is some text written to the file.\n";
outputFile << "Another line of text.\n";
```

- In this example, we declare an object of the ofstream class called outputFile, open the file "filename.txt" for writing, and then write two lines of text to the file using the << operator.
- Note that you should include the newline character (\n) at the end of each line if you want the lines to be separated by newlines in the file.

Step 5: Close the file using the close() method of the output file stream object.

- Once you're done writing to the file, you should close it by calling the close() method on the output file stream object, like this:

```
outputFile.close();
```

- This ensures that any data remaining in the output buffer is written to the file and that the file is properly closed.

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
ofstream myfile;
myfile.open("firstfile.txt");
 myfile<<"Samuel Lenjissa\n";
 myfile<<"Jalane Tola";
 myfile.close();
}
```

- The code is create a file named "firstfile.txt" and writes two lines of text into it: "Samuel Lenjissa" and "Jalane Tola".

Cont..

- there are a couple of things you could improve:

1. You should check if the file was opened successfully before writing to it.You can do this by calling the is_open() method on the myfile object after calling open(), and output an error message if the file couldn't be opened.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream myfile("firstfile.txt");
    if (myfile.is_open()) {
        myfile << "Samuel Lenjissa\n";
        myfile << "Jalane Tola\n";
        myfile.close();
        cout << "File written successfully.\n";
    } else {
        cout << "Error opening file.\n";
    }
    return 0;
}
```

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

int main() {
    // Open the output file
    ofstream outputFile("output.txt");

    // Write the result of the process to the output file
    int result = 42;
    outputFile << "The result of the process is: " << result << endl;

    // Close the output file
    outputFile.close();

    return 0;
}
```

Structure and file

```
#include <iostream>
#include <fstream>
using namespace std;
// Define a struct to represent a person
struct Person {
   string name;
   int age;
};
int main() {
   // Open the output file
   ofstream outputFile("output.txt");
   // Create a Person struct and write it to the output file
   Person p = {“Tola", 30};
   outputFile << "Name: " << p.name << ", Age: " << p.age << endl;
   // Close the output file
   outputFile.close();
   return 0;
}
```

Array of structure to file

```
#include <iostream>
#include <fstream>
using namespace std;
// Define a struct to represent a person
struct Person {
   string name;
   int age;
};
int main() {
   // Open the output file
   ofstream outputFile("output.txt");
   // Create an array of Person structs and write each one to the output file
   Person people[] = {{"John", 30}, {"Jane", 25}, {"Bob", 40}};
   int numPeople = sizeof(people) / sizeof(Person);
   for (int i = 0; i < numPeople; i++) {
      outputFile << "Name: " << people[i].name << ", Age: " << people[i].age << endl;
   }
   // Close the output file
   outputFile.close();
   return 0;
}
```

File and function

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

// Define a function to write a string to a file
void writeToFile(string filename, string text) {
   ofstream outputFile(filename);
   if (outputFile.is_open()) {
      outputFile << text << endl;
      outputFile.close();
      cout << "File written successfully.\n";
   } else {
      cout << "Error opening file.\n";
   }
}

// Define a function to read a string from a file
string readFromFile(string filename) {
   ifstream inputFile(filename);
   if (inputFile.is_open()) {
      string text;
      getline(inputFile, text);
      inputFile.close();
      return text;
   } else {
      cout << "Error opening file.\n";
      return "";
   }
}

int main() {
   // Write a string to a file using the writeToFile
function
   writeToFile("output.txt", "Hello, world!");

   // Read a string from a file using the readFromFile
function
   string text = readFromFile("input.txt");
   cout << "Text from file: " << text << endl;

   return 0;
}
```

Read a File

1. Include the <fstream> header to use file stream objects.
2. Declare an input file stream object, such as ifstream inputFile.
3. Open the file using the open() method of the input file stream object, passing the filename as an argument.
4. Check if the file was opened successfully using the is_open() method of the input file stream object.
5. Read data from the file using input stream operators (>> for formatted input or getline() for unformatted input).
6. Close the file using the close() method of the input file stream object.

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

int main() {
   // Declare an input file stream object and open the file
 ifstream inputFile("input.txt");
   if (inputFile.is_open()) {
      // Read a single line of text from the file
      string text;
      getline(inputFile, text);
      cout << "Text from file: " << text << endl;

      // Close the file
      inputFile.close();
   } else {
      cout << "Error opening file.\n";
   }

   return 0;
}
```

- In this example, we declare an input file stream object called inputFile and open the file "input.txt" for reading.
- We then check if the file was opened successfully using the is_open() method of the input file stream object.
- If it was, we read a single line of text from the file using the getline() function and store it in a string called text.
- We output the text to the console using cout.
- Finally, we close the file using the close() method of the input file stream object.
- Note that you can read data from the file using input stream operators (>>) as well, but getline() is often more convenient for reading entire lines of text.

- If you want to read multiple lines of text from a file in C++, you can use a loop to read each line of text one at a time.

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

int main() {
   // Declare an input file stream object and open the file
   ifstream inputFile("input.txt");
   if (inputFile.is_open()) {
      // Read each line of text from the file and output it to the console
      string line;
      while (getline(inputFile, line)) {
         cout << "Line from file: " << line << endl;
      }

      // Close the file
      inputFile.close();
   } else {
      cout << "Error opening file.\n";
   }

   return 0;
  }
```

Cont..

- In this example, we declare an input file stream object called inputFile and open the file "input.txt" for reading.
- We then check if the file was opened successfully using the is_open() method of the input file stream object.
- we use a loop to read each line of text from the file using the getline() function and store it in a string called line.
- We output each line of text to the console using cout. Finally, we close the file using the close() method of the input file stream object.

Binary file

- In C++, there are two types of files: text files and binary files.
- Text files are files that contain human-readable text, while binary files are files that contain non-textual data, such as images, audio, or serialized objects.

Cont..

- Here are the differences between text and binary files in C++:
- Text files are stored as a sequence of characters, with each character encoded using a specific character encoding (such as ASCII or UTF-8).
- Text files can be opened and read using input file stream objects (ifstream) and written to using output file stream objects (ofstream).
- Binary files are stored as a sequence of bytes, with each byte representing a specific value in the file.
- Binary files can be opened and read using input binary file stream objects (ifstream with the **ios::binary flag set**) and written to using output binary file stream objects (ofstream with the ios::binary flag set).

example

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

int main() {
   // Write an integer to a binary file
   int number = 42;
   ofstream outputFile("output.bin",
ios::binary);
   if (outputFile.is_open()) {
      outputFile.write((char*)&number,
sizeof(number));
      outputFile.close();
      cout << "Binary file written
successfully.\n";
   } else {
      cout << "Error opening file.\n";
   }
   // Read an integer from a binary file
   int readNumber;
   ifstream inputFile("output.bin",
ios::binary);
   if (inputFile.is_open()) {
      inputFile.read((char*)&readNumber,
sizeof(readNumber));
      inputFile.close();
      cout << "Number read from file: " <<
readNumber << endl;
   } else {
      cout << "Error opening file.\n";
   }

   return 0;
}
```

Cont..

- In this example, we write an integer value of 42 to a binary file called "output.bin".
- We use an output binary file stream object (ofstream) with the ios::binary flag set to open the file for writing.
- We then use the write() function to write the integer value to the file, casting the address of the number variable to a char* to ensure that the bytes are written correctly.

Cont..

- To read the integer value from the binary file, we declare another integer variable called readNumber, and read from the binary file using an input binary file stream object (ifstream) with the ios::binary flag set.
- We use the read() function to read the integer value from the file, casting the address of the readNumber variable to a char* to ensure that the bytes are read correctly.
- Finally, we output the value of readNumber to the console using cout.

Cont..

- In the line outputFile.write((char*)&number, sizeof(number));
- we are using the write() function to write binary data to the output file stream object outputFile.
- The first argument to write() is a pointer to the buffer containing the data to be written.
- In this case, we are passing a pointer to the number variable, which is an integer value of 42.
- The second argument to write() is the number of bytes to write to the file.
- In this case, we are passing sizeof(number), which is the size of the number variable in bytes. On most systems, an int variable is 4 bytes in size, so sizeof(number) will evaluate to 4.

Cont..

- When we cast the address of the number variable to a char* using (char*)&number, we are effectively telling the compiler to treat the memory location of the number variable as a sequence of bytes.
- In C++, every variable is allocated a unique memory address where its value is stored. T
- he memory address is represented as a hexadecimal number, which is a base-16 number system. For example, the memory address of the number variable might be 0x7ffeedb8 (this value will be different on different systems).
- When we cast the address of number to a char*, we are telling the compiler to treat this memory address as a pointer to a char variable, rather than an int variable.
- This means that the compiler will interpret the binary data starting at this memory address as a sequence of char values, rather than int values.

Cont..

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   // Write a string to a binary file
   string message = "hello";
   ofstream outputFile("output.bin", ios::binary);
   if (outputFile.is_open()) {
      outputFile.write(message.c_str(), message.length());
      outputFile.close();
      cout << "Binary file written successfully.\n";
   } else {
      cout << "Error opening file.\n";
   }
   return 0;
}
```

Random Access Files

- A random access file in C++ is a type of file that allows you to read and write data to any location within the file.
- This is different from a sequential access file, which only allows you to read or write data in a sequential order from the beginning of the file to the end.
- To work with a random access file in C++, you can use the fstream library.

Random file access with seekg() and seekp()

- Here are the basic steps for working with a random access file:

1. Open the file using fstream with the appropriate mode (ios::in, ios::out, or ios::in | ios::out).
2. Use the **seekg()** function to move the file pointer to the desired location for reading.
3. read data from the file.
4. Use the **seekp()** function to move the file pointer to the desired location for writing.
5. Write data to the file using the << operator or the write() function.
6. Close the file using the **close()** function.

write data to the file using **seekp()** function .

- function take two arguments:
- the first argument is the number of bytes to offset from the position specified by the second argument.

second argument can be one of three constants:

✓ios::beg (beginning of the file)

✓ios::cur (current position of the file pointer)or

✓ios::end (end of the file).

Ios seek flag	Meaning
beg	The offset is relative to the beginning of the file (default)
cur	The offset is relative to the current location of the file pointer
end	The offset is relative to the end of the file

- For example, to set the file pointer to the beginning of a file, you can use:

```
fstream file("example.txt");
file.seekp(0, ios::beg);
```

✓ This will set the file pointer to the beginning of the file "example.txt".

✓ the first argument is 0, which specifies the number of bytes to offset from the beginning of the file.

✓ and the second argument is ios::beg, which specifies that the offset should be relative to the beginning of the file.

Example 2: to set the file pointer to a specific position in a file for output operations, you can use:

```
ofstream file("example.txt");
file.seekp(10, ios::beg);
```

- ✓ This will set the file pointer to 10 bytes from the beginning of the file "example.txt".

Example 3: **move from current position.**

- ✓ If you have a file pointer that is currently pointing to the 10th byte in a file, you can use **seekg(5, ios::cur)** to move the file pointer 5 bytes forward, so that it points to the 15th byte in the file.

Exsmple 4: move from end posotion.

- ✓ For example, if you have a file that is 100 bytes long, you can use **seekg(-10, ios::end**) to move the file pointer 10 bytes backwards from the end of the file, so that it points to the 90th byte in the file.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   // Open file for output
   ofstream file("example.txt",ios::out);
   if (!file.is_open()) {
      cout << "Failed to open file\n";
      return 1;
   }
file << "0123456789";
   // Move the file pointer 5 bytes forward
from the beginning of the file
   file.seekp(5, ios::beg);
   // Write some more data to the file
   file.write("abc", 3);
// Move the file pointer 2 bytes backwards
from the current position
   file.seekp(-2, ios::cur);
   // Write some more data to the file
   file<<"def";
   // Move the file pointer 3 bytes
backwards from the end of the file
   file.seekp(-3, ios::end);
   // Write some more data to the file
   file<<"ghi";
   // Close the file
   file.close();
   // Open file for input
   ifstream input("example.txt", ios::in);
   if (!input.is_open()) {
      cout << "Failed to open file\n";
      return 1;
   }
   // Read the entire contents of the file
   string buffer;
   getline(input,buffer);
   cout << buffer << endl;
   // Close the file
   input.close();
   return 0;
}
```

Description of example

- In the above example, we first open the file "example.txt" for output and write some data to it.
- We then use seekp() to move the file pointer 5 bytes forward from the beginning of the file, write some more data.
- Then move the file pointer 2 bytes backwards from the current position, write some more data.
- And move the file pointer 3 bytes backwards from the end of the file, write some more data.
- Finally, we close the file and open it again for input.
- We read the entire contents of the file into a buffer and print it to the console. The output should be:

```
01234adghi
```

read data from the file using **seekg()** function .

- o read data from a specific position in a file using the seekg() function, you can follow these steps:

1. Open the file in input mode using **ifstream.**
2. Use the **seekg()** function to set the position of the file pointer to the desired location.
3. Read data from the file using the >> operator, getline(), or the read() function.

Cont..

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   ifstream file("example.txt", ios::in);
   if (!file.is_open()) {
      cout << "Failed to open file\n";
      return 1;
   }
   // Read some data from the beginning
of the file
   file.seekg(0);
   string line;
   getline(file, line);
   cout << "Line 1: " << line << '\n';

   // Read some data from the middle of
the file
   file.seekg(-5, ios::cur);
   char buffer[4];
   file.read(buffer, 3);
   buffer[3] = '\0';
   cout << "Middle: " << buffer << '\n';

   // Read some data from the end of the
file
   file.seekg(-9,ios::end);
   int number;
   file >> number;
   cout << "End: " << number << '\n';

   file.close();
   return 0;
}
```

Example description

- In the above example, we first open the file "example.txt".
- We then use seekg() to read data from the beginning, middle, and end of the file.
- Note that we use ios::cur as the second argument to seekg() when we want to move the file pointer relative to its current position.
  - ✓ This allows us to seek to a specific position in the middle of the file.
- We also use ios::end as the second argument to seekg() when we want to move the file pointer relative to the end of the file.