Advanced Programming

Files and Streams in C++

Dr. Zeyad Safaa Younus Saffawi

Why Files?

- Store data permanently
- Handle large amounts of data
- Used in real applications
- Work with logs and reports

FILES AND STREAMS in C++

In previous programs, we used the **iostream standard library <iostream>**, which provides the **cin** for reading from the keyboard and **cout** for writing to the screen.
However, real-world applications require **storing** and **retrieving** data from **files** instead of just the console. For this purpose, C++ provides another standard library to **read from a file or write to a file** called:

```
#include <fstream>
```

- **File Stream Data Types**
- The **<fstream>** library defines three important data types.

Data Type	Description
ofstream	Output file stream used to create files and write data to files.
ifstream	Input file stream used to read data from files.
fstream	General file stream used for both reading and writing.

- **Required Header Files**
- To perform file processing in C++, the following headers must be included:

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

Opening a File

- Before performing any read or write operation, a file must be **opened first**.
- C++ provides different file stream classes:
  - **ofstream** → used to **open** files for **writing** to files
  - **ifstream** → used to **open** files for **reading from files**
  - **fstream** → used for **both reading and writing**
- **Syntax of open a file**

1. Constructor Syntax: `ofstream objectName("filename", mode);`

Example: `ofstream file ("data.txt");`

2. **Function Syntax:** `file.open("filename", ios::mode);`

Example

```
ofstream file;
file.open("data.txt", ios::out);
```

Parameters

1. **filename** → the name of the file to open
2. **mode** → specifies how the file will be opened (read, write, append, etc.)

File Modes

- **File Opening Modes:**
- File modes determine how a file is opened and used.

Mode	Description
ios::app	Append mode (data is added to the end of the file).
ios::ate	Open file and move pointer to the end.
ios::in	Open file for reading.
ios::out	Open file for writing.
ios::trunc	delete file contents if the file already exists.
ios::binary	binary mode

- Modes can be **combined using OR operator |**.
- Example:

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc);
```

This means:

- open file for writing
- if file already exists → delete old content
- if file does not exist → create new file

File Modes

- Delete old contents
  - ofstream file1("data.txt", ios::out | ios::trunc);

- Add content at end of the file
  - ofstream file2("data.txt", ios::out | ios::app);

Example:
writing to a file

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream file;
    file.open("data.txt", ios::out);
    file << "Hello World";
    file.close();
    return 0;
}
```

Example: reading from a file

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream file;
    file.open("data.txt", ios::in);
    string text;
    file >> text;
    cout << text;
    file.close();
    return 0;
}
```

Opening File for Reading and Writing

1. Opening File for Reading and Writing

To open a file for both reading and writing, we use fstream:

```
fstream outfile;
outfile.open("file.dat", ios::out | ios::in);
```

- **This allows:**
  - **reading from the file**
  - **writing to the same file**

2. Closing a File

- When a program **ends**, C++ automatically **closes files**.
  However, it is **good programming practice** to close files manually.

Syntax

```
file-name.close();
```

Example:

```
outfile.close();
```

Example: reading from a file and writing to it

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    fstream file;
    file.open("data.txt", ios::in | ios::out | ios::trunc);
    file << "Hello";
    file.seekg(0); // move back to start
     string text;
    file >> text;
    cout << text;

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

Writing to a File in C++

- Writing to a file is done using the **stream insertion operator** <<, just like using **cout** to display output on the screen.

Example:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ofstream outfile;
   outfile.open("data.txt");
   outfile << "Hello" << endl;
   outfile << 25 << endl;
   outfile.close();
   return 0;
}
```

Writing to a File in C++

Important Note (File Overwriting)

By default, ofstream opens a file using **ios::out mode**.

- This means:
  - The existing content of the file will be **deleted (overwritten)**
- **Example (Problem)**

```
ofstream file("data.txt");
file << "New Data";
```

- This will **erase old data** in the file.

- **Solution (Append Mode)**
  - To keep existing data, use **ios::app**:

```
ofstream file("data.txt", ios::app);
file << "New Data";
```

- This will **add data to the end of the file** without deleting previous content.
- **Real-World Example**
  - Storing student records:

```
ofstream file("students.txt", ios::app);
file << "Ali 90" << endl;
```

- This ensures that new records **are added without removing existing ones**.

Reading from a File

- Reading from a file is done using the **stream extraction operator >>**, just like using **cin** to **read input** from the keyboard.

Example:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream infile;
    infile.open("data.txt");
    string name;
    infile >> name;
    cout << name;
    infile.close();
    return 0;
}
```

Reading from a File

Important Note

The operator >> reads **only one word** (until space or newline).

- This means:
  - It will NOT read full sentences with spaces
- **Example of the Problem**
  - If the file contains: Ali Ahmed

```
infile >> name;
```

- Output will be: Ali
- **Solution (Read Full Line)**
  - To read a complete line, use getline():

```
string line;
getline(infile, line);
```

  - This reads the entire line including spaces.
- **Real-World Example**

```
ifstream file("students.txt");
string name;
getline(file, name);
cout << name;
```

Reading Until End of File (EOF)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream file("data.txt");

    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }
    file.close();
}
```

- Reads the file **line by line**
- Stops automatically at **End Of File (EOF)**

Additional Functions Used

- C++ provides additional functions to handle input more effectively.

1. getline()

- Used to read a **full line of text including spaces**.
- Syntax : getline(stream, variable);

Example:

```
string data;
getline(cin, data);
```

- Reads the entire line from the user (including spaces)
- **Reading from a file**

```
ifstream file("data.txt");
string line;
getline(file, line);
```

- Reads a full line from a file

Additional Functions Used

2. ignore()

- Used to **ignore unwanted characters in the input buffer**

```
cin.ignore();
```

  - Skips leftover characters (like newline \n)
- Notes:
  - When mixing cin >> with getline():

```
int age;
cin >> age;
cin.ignore();   // important
string name;
getline(cin, name);
```

- Without ignore(), getline() may be skipped!

Additional Functions Used

Example with a file:

```
ifstream file("data.txt");
int num;
file >> num;
file.ignore();
string line;
getline(file, line);
```

Example: Reading and Writing

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
   char data[100]; int data1;
   // open file for writing
   ofstream outfile;
   outfile.open("afile.dat");
   cout << "Writing to a file" << endl;
   cout << "Enter your name: ";
   cin.getline(data, 100);
   outfile << data << endl;
   cout << "Enter your age: ";
   cin >> data1;
   cin.ignore();
   outfile << data1 << endl;
   outfile.close();
   // open file for reading
   ifstream infile;
   infile.open("afile.dat");
   cout << "Reading from a file" << endl;
   infile >> data;
  cout << data << endl;
   infile >> data1;
   cout << data1 << endl;
   infile.close();
   return 0;
}
```

Output:

```
Writing to a file
Enter your name: Ali saad
Enter your age: 9

Reading from a file
Ali saad
9
```

File Position Pointers

- C++ allows controlling the position of the read |write pointer inside a file. This is useful for **random access** instead of reading sequentially.

Function	Used With	Purpose
seekg()	istream	**move read pointer in a file**
seekp()	ostream	**move write pointer in a file**

- **Seek Directions**

Direction	Meaning
fileObject.seekg(n, **ios::beg**);	Move to a specific (nth) byte from the **beginning of file**
fileObject.seekg(n, **ios::cur**);	Move n bytes forward from **current position**
fileObject.seekg(n, **ios::end**);	Move n bytes before the **end of file**
fileObject.seekg(0, **ios::end**);	Move to the end of file

Example

```
ifstream file("data.txt");
file.seekg(0, ios::beg);  // go to start
string data;
getline(file, data);
cout << data;
file.close();
```

Example : Write Data to a File
Write a program to read a student's **name and age** from the user and store them in a file called **student.txt**.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   char name[50];
   int age;
   ofstream outfile;
   outfile.open("student.txt");
   cout << "Enter name: ";
   cin >> name;
   cout << "Enter age: ";
   cin >> age;
   outfile << name << endl;
   outfile << age << endl;
   outfile.close();
   cout << "Data written to file successfully.";
   return 0;
}
```

Output:
Enter name: Mohammed
Enter age: 18
Data written to file successfully.

Example : Read Data from a File
Write a program to **read a name and age from the file student.txt** and display them on the screen.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   char name[50];
   int age;
   ifstream infile;
   infile.open("student.txt");
   infile >> name;
   infile >> age;
   cout << "Name: " << name << endl;
   cout << "Age: " << age << endl;
   infile.close();
   return 0;
}
```

Example : Write Multiple Numbers and Read Them

Write a program to **store 5 numbers in a file called numbers.txt** and then read them and print them.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   int num, i;
   ofstream outfile("numbers.txt");
   cout << "Enter 5 numbers:\n";
   for(i = 0; i < 5; i++)
   {
      cin >> num;
      outfile << num << endl;
   }
   outfile.close();
   ifstream infile("numbers.txt");
   cout << "Numbers in file:\n";
   while(infile >> num)
   {
      cout << num << endl;
   }
   infile.close();
   return 0;
}
```

Example: Read and Write Data Using File

Write a C++ program to:

Read **name and age** from the user.

Store them in a file called **data.txt**.

Close the file.

Open the same file again.

Read the data from the file and print it on the screen.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   char name[50];
   int age;
   // Writing to file
   ofstream outfile;
   outfile.open("data.txt", ios::out);
   cout << "Enter your name: ";
   cin.getline(name,50);
   cout << "Enter your age: ";
   cin >> age;
   outfile << name << endl;
   outfile << age << endl;
   outfile.close();
   // Reading from file
   ifstream infile;
   infile.open("data.txt", ios::in);
   cout << "\nReading from file:\n";
   infile.getline(name,50);
   infile >> age;
   cout << "Name: " << name << endl;
   cout << "Age: " << age << endl;
   infile.close();
   return 0;
}
```

References

- Gaddis, T. (2014). *Starting out with C++: From control structures through objects* (8th ed.). Pearson.
- Soulié, J. (2007, April 24). C++ language tutorial. cplusplus.com.
- Tutorials Point. (n.d.). *Learn C++ programming language*. Tutorials Point.