The Zen of C++

2nd Edition

Adnan Zejnilovic

Chapter 5

Streams – Input, Output, Customization

Streams

- Streams are buffers that can hold data
- A process produces data and “parks” it in a stream for further processing
- There are input and output streams
- Input streams
  - Data entered via keyboard
  - File from the disk
  - Data from the network
- Output Streams
  - Data that needs to be written to a screen, printer,

Extracting Data From a Stream

- Your program views the data in a stream as series of bytes
- Each byte is interpreted as a character
- The *stream extraction operator* “>>” translates the data to appropriate data type as requested by the **cin** object:

```
int myNumber;
cout << "Enter a number: ";
cin >> myNumber;
```

myNumber will be populated with an integer

Writing Data to a Stream

- The data is stored in binary format in the RAM
- It gets converted to characters before it is “written” to the stream
- This process is the same whether the program is interacting with the keyboard buffer or a file stream

File Types

- Files are used to store data between program runs
- Files can be binary or text files

Text Files	Binary Files
Store data in text (character) format	Store data in binary (raw) format
The data from the RAM is translated to text before it is stored in a text file	The data from the RAM is not translated to text before it is stored in a binary file
Can be read by humans	Cannot be read by humans

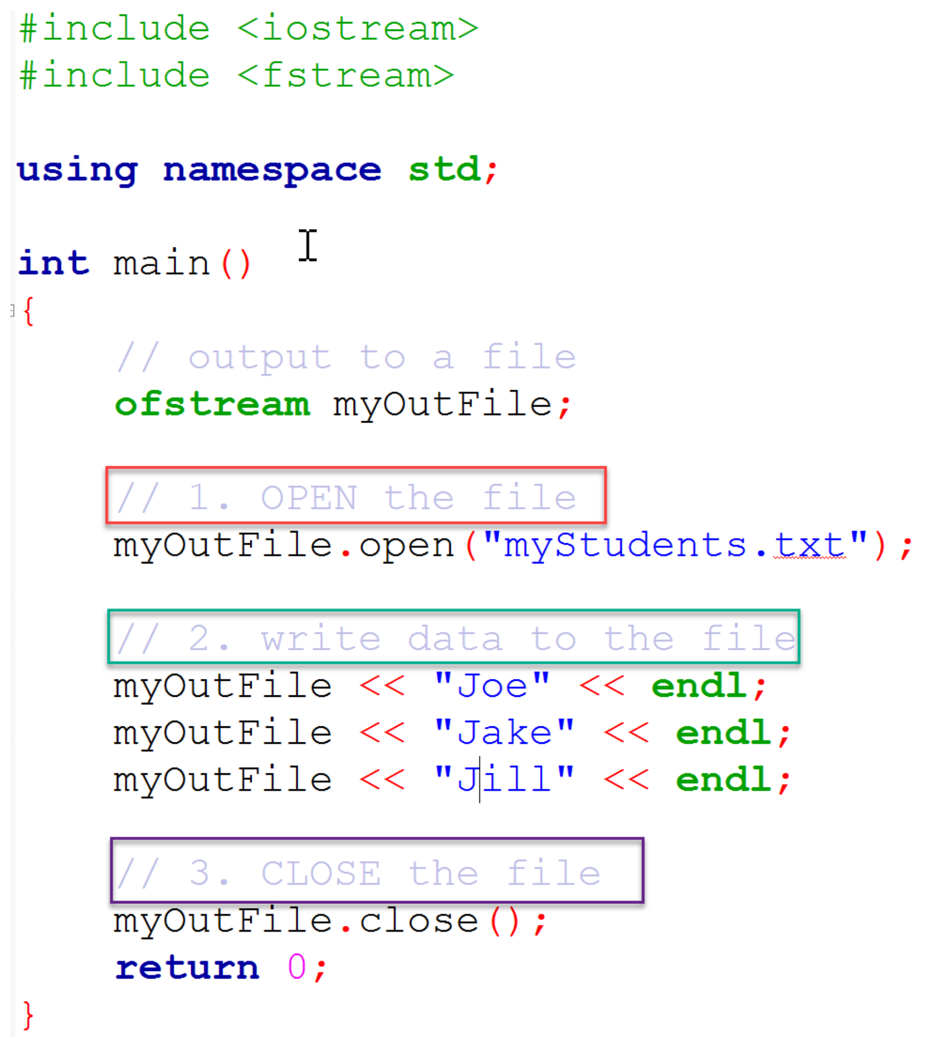
Reading From and Writing To Files Programmatically

- Must include the <fstream> header
- A C++ program must specify how it intends to use a file
- If your program is going to read data from a file, then you need to declare a variable of data type ***ifstream***
- If your program is going to write data to a file, then you need to declare a variable of data type ***ofstream***

Writing to a Text File

Steps:

- Declare a variable of data type ofstream
- Open the file
- Write data to the file
- Close the file



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

using namespace std;

int main()
{
    // output to a file
    ofstream myOutFile;

    // 1. OPEN the file
    myOutFile.open("myStudents.txt");

    // 2. write data to the file
    myOutFile << "Joe" << endl;
    myOutFile << "Jake" << endl;
    myOutFile << "Jill" << endl;

    // 3. CLOSE the file
    myOutFile.close();
    return 0;
}
```

Writing to a Text File

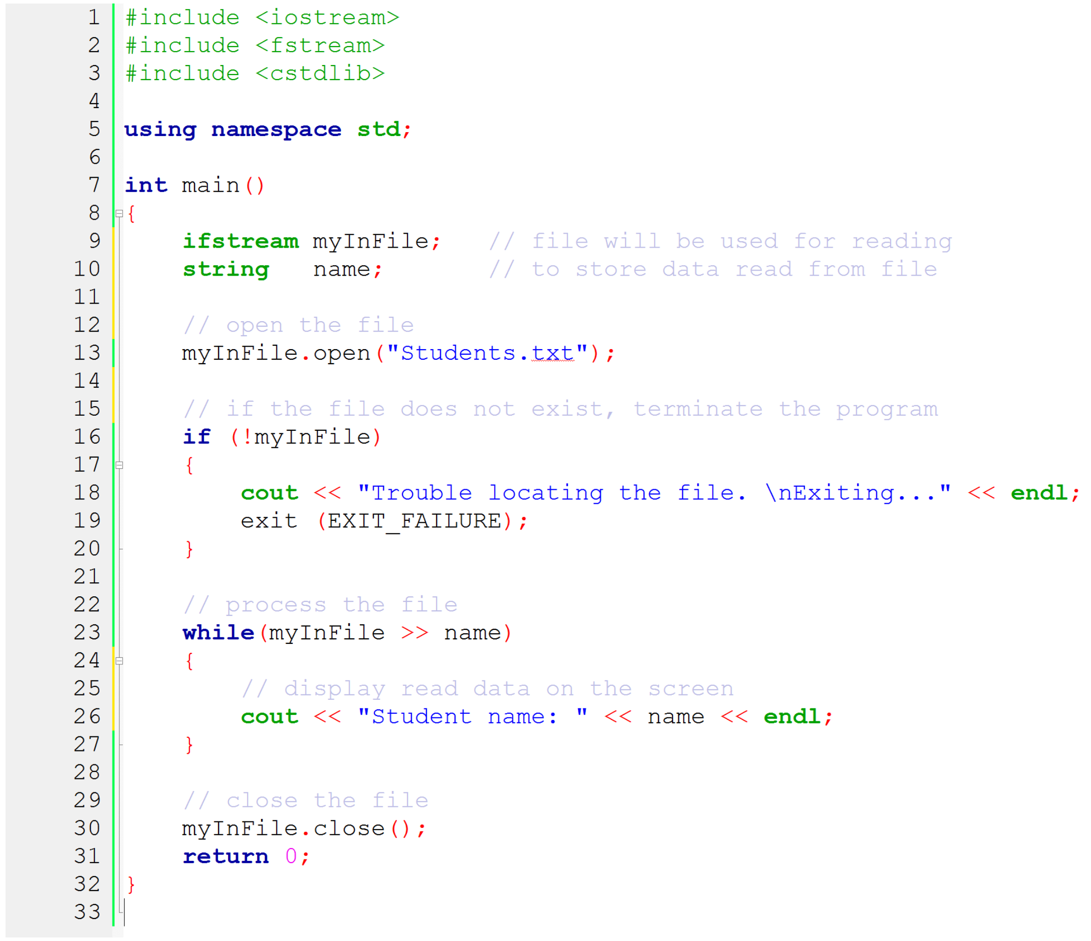
- When opening a file for output (“writing to a file”), it is not necessary to check whether the file exists
- If it does not exist, it will be created
- If it does exist, the information in it will be erased
- It is possible to append new data to existing data in a file
- Also, if you specify only the file name in the string literal parameter passed to the open method of ofstream object, the file will be created in the project directory:

```
// 1. OPEN the file
myOutFile.open("myStudents.txt");
```

- You could specify full path to a differnet directory if you need to

Reading From a File

- Accomplished with the aid of the stream extraction operator (>>)
- The file MUST EXIST in the directory, otherwise the program may crash
- Thus, you need to check if the file opened successfully before proceeding



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

using namespace std;

int main()
{
    ifstream myInFile;    // file will be used for reading
    string   name;        // to store data read from file

    // open the file
    myInFile.open("Students.txt");

    // if the file does not exist, terminate the program
    if (!myInFile)
    {
        cout << "Trouble locating the file. \nExiting..." << endl;
        exit (EXIT_FAILURE);
    }

    // process the file
    while(myInFile >> name)
    {
        // display read data on the screen
        cout << "Student name: " << name << endl;
    }

    // close the file
    myInFile.close();
    return 0;
}
```

Reading From a File

- If the data stored in the file contains white space, then the stream extraction operator is not going to succeed in reading the data
- Alternatives:
  - ***`fileHandle.get(ch)`*** where ***`ch`*** is a character data type and ***`fileHandle`*** is an instance of the ***`ifstream`*** object. The get function reads one character at a time
  - ***`getline(fileHandle, line)`*** where ***`fileHandle`*** is an instance of the ***`ifstream`*** object and ***`line`*** is an instance of the ***`string`*** object.
  - If the data in the file is separated by comas, then the following function ***`getline(fileHandle, line, ',')`*** is the proper way to read it
    - ***`fileHandle`*** is an instance of the ***`ifstream`*** object and
    - ***`line`*** is an instance of the ***`string`*** object.
    - *`','`* is a delimiter.