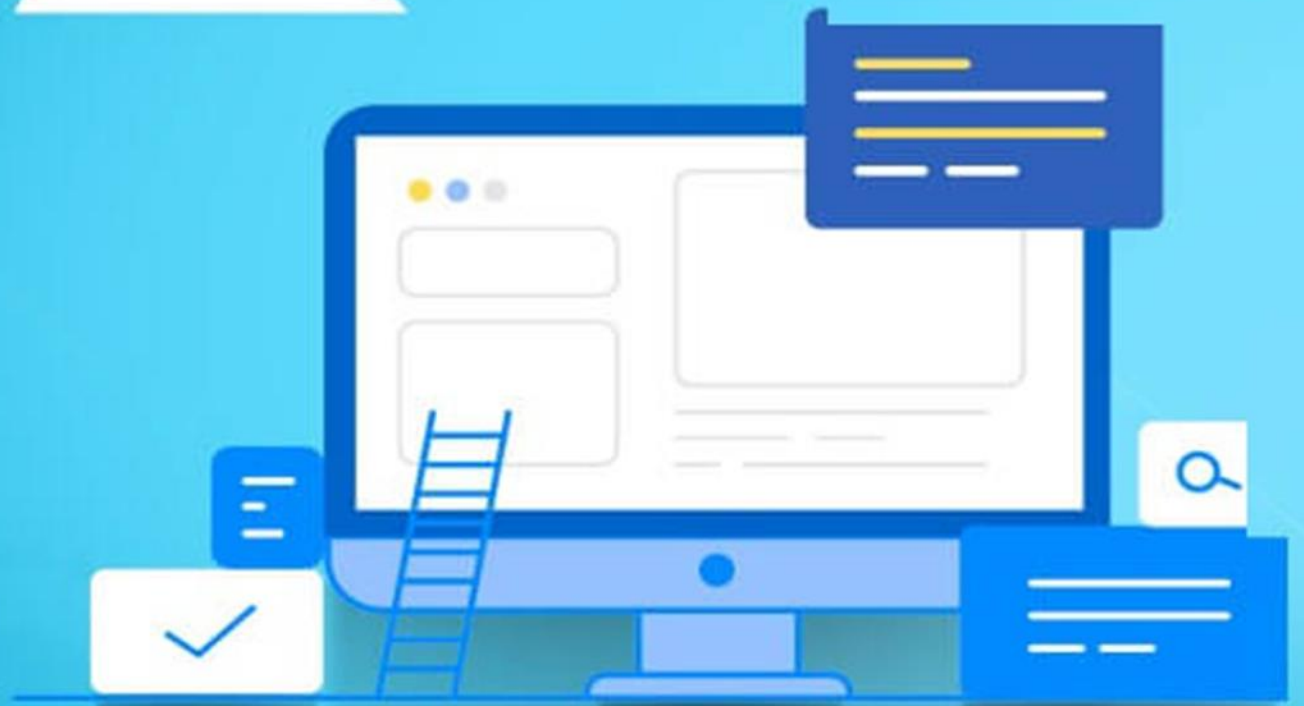
C++ Stream



By – Mohammad Imran

Concept of Streams

Stream in C++ means a stream of characters that gets transferred between the program thread and input or output. There are a number of C++ stream classes eligible and defined which is related to the files and streams for providing input-output operations. All the classes and structures maintaining the file and folders with hierarchies are defined within the file with iostream.h standard library. Classes associated with the C++ stream include ios class, istream class, and ostream class. Class ios is indirectly inherited from the base class involving iostream class using istream class and ostream class which is declared virtually.

By – Mohammad Imran

Concept of Streams

In C++, the concept of **streams** refers to a continuous flow of data, which can be input to or output from a program. Streams provide a high-level abstraction for handling input and output operations, allowing developers to read from and write to various data sources (like files, console, or network) in a uniform way.

Stream Characteristics

Input and Output:

- ✓ **Input Streams:** These are used for reading data from a source, such as a keyboard (standard input) or a file. The primary input stream in C++ is **std::cin**.
- ✓ **Output Streams:** These are used for writing data to a destination, such as a screen (standard output) or a file. The primary output stream in C++ is **std::cout**.

Stream Characteristics

Stream Classes:

C++ provides several predefined stream classes, including:

- ✓ **std::istream**: Base class for input streams.
- ✓ **std::ostream**: Base class for output streams.
- ✓ **std::fstream**: A class that handles both input and output operations on files.
- ✓ **std::ifstream**: For input file streams.
- ✓ **std::ofstream**: For output file streams.

Stream Characteristics

Buffering:

- ✓ Streams use buffering to improve performance. Data is read or written in blocks rather than one byte at a time, which reduces the number of I/O operations.

Type Safety:

- ✓ C++ streams provide type safety, allowing for easy formatting and conversion of various data types. For example, you can easily read an integer or a string from an input stream.

Stream Characteristics

Manipulators:

- ✓ Streams support manipulators, which are special functions that modify the format or behavior of I/O operations. Common manipulators include **`std::endl`** (to insert a newline) and **`std::setw`** (to set field width).

Stream State:

- ✓ Streams maintain a state (good, bad, eof, fail) that indicates whether the last operation was successful. This can be checked using member functions like **`good()`**, **`eof()`**, **`fail()`**, and **`clear()`**.

By – Mohammad Imran

Stream

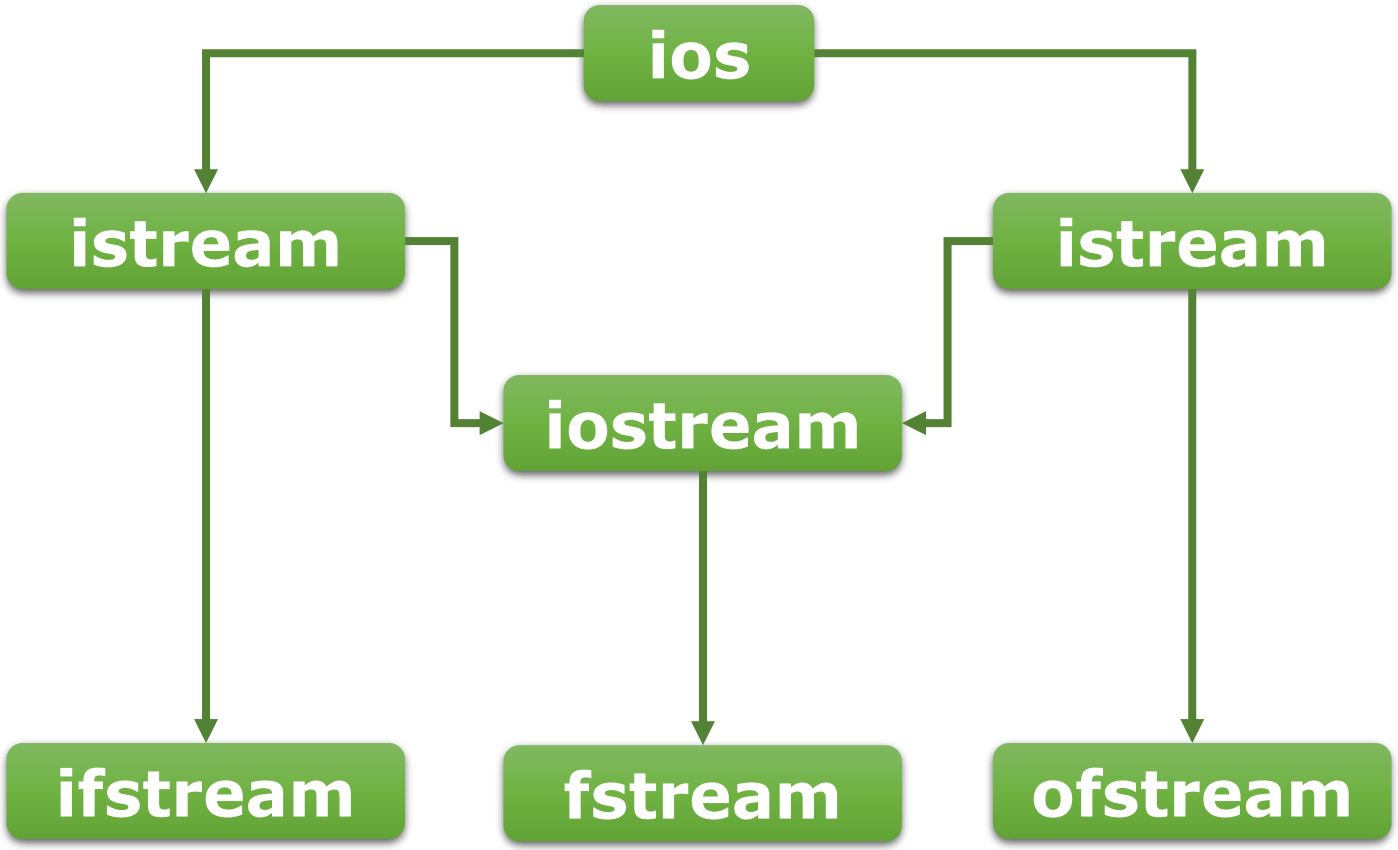
Class Hierarchy

There is a number of stream classes in the hierarchy which is defining and giving different flows for the varied objects in the class. The hierarchy is maintained in a way where it gets started from the top class which is the ios class followed by all the other classes involving istream class, ostream class, iostream class, istream_withassign class, and ostream_withassign class. The iosclass in the hierarchy is the parent class which is considered as a class from where both the istream and ostream class gets inherited. Both the istream class and ostream class constitute the ios class which is the highest level of the entire hierarchy of C++ stream classes.

By – Mohammad Imran

Stream Class Hierarchy

The figure below shows the hierarchy of the stream class library:



istream Class

`istream` being a part of the ios class which is responsible for tackling all the input stream present within the stream. It provides all the necessary and important functions with the number of functions for handling all the strings, chars, and objects within the istream class which comprises all these functions such as **`get`**, **`read`**, **`put`**, etc.

By – Mohammad Imran

istream Class

Example

```
#include <iostream>
using namespace std;
int main()
{
   char p;
   cin.get(p);
   cout << p;
   return 0;
}
```

OUTPUT

```
A
A
```

By – Mohammad Imran

ostream Class

This class as part of the ios class is also considered as a base class that is responsible for handling output stream and provides all the necessary functions for handling chars, strings, and objects such as **`put`**, **`write`**, etc.

ostream Class

Example

```
#include <iostream>
using namespace std;
int main()
{
   char r_t;
   cin.get(r_t);
   cout.put(r_t);
   return 0;
}
```

OUTPUT

```
Z
Z
```

By – Mohammad Imran

iostream Class

`iostream` class is the next hierarchy for the ios class which is essential for input stream as well as output stream because **`istream`** class and **`ostream`** class gets inherited into the main base class. As the name suggests it provides functionality to tackle the objects, strings, and chars which includes inbuild functions of **`put`**, **`puts`**, **`get`**, etc.

By – Mohammad Imran

iostream Class

Example

```
#include <iostream>
using namespace std;
int main()
{
   char str[15] = "Education Portal";
   cout.write(str, 9);
   return 0;
}
```

OUTPUT

```
Education
```

By – Mohammad Imran

ios Class

`ios` class is the highest class in the entire hierarchical structure of the C++ stream. It is also considered as a base class for **`istream`**, **`stream`**, and **`streambuf`** class. It can be said that the ios class is basically responsible for providing all the input and output facilities to all the other classes in the stream class of C++.

By – Mohammad Imran

ios Class

Example

```
using namespace std;
#include <iostream>
int main()
{
   int n;
   cin >> n;
   cout << n;
   return 0;
}
```

OUTPUT

```
100
100
```

istream_withassign Class

This class is considered as a variant for the **`istream`** class that provides the class privilege for the class to assign object. The predefined object which can be called a build in the function of this class is used which is responsible for providing getting the stream facility and thus allows the object to reassign at the run time for different stream objects. **`istream_withassign`** class acts as the primary class for the other classes as part of the **`istream`** class.

By – Mohammad Imran

istream_withassign Class

Example

```
#include <iostream>
using namespace std;
int main()
{
   char istream_withassign[8];
   cin.get(istream_withassign, 8);
   cout << istream_withassign << '\n';
   cin.get(istream_withassign, 8);
   cout << istream_withassign << '\n';
   return 0;
}
```

OUTPUT

```
I am fine
I am fi
ne
```

By – Mohammad Imran

ostream_withassign Class

This class is responsible for providing object assigned to the class and is considered as a variant itself for the **`ostream`** class of the C++ stream. All the build-in functions such as **`cout`**, **`cerr`**, **`clog`** is the already present objects of the same class and are reassigned at execution time for the different **`ostream`** object.

By – Mohammad Imran

ostream_withassign Class

Example

```
using namespace std;
#include <iostream>
int main()
{
   char ostream_withassign[10];
   cin.get(ostream_withassign, 10);
   cout << ostream_withassign << '\n';
   cin.get(ostream_withassign, 10);
   cout << ostream_withassign << '\n';
   return 0;
}
```

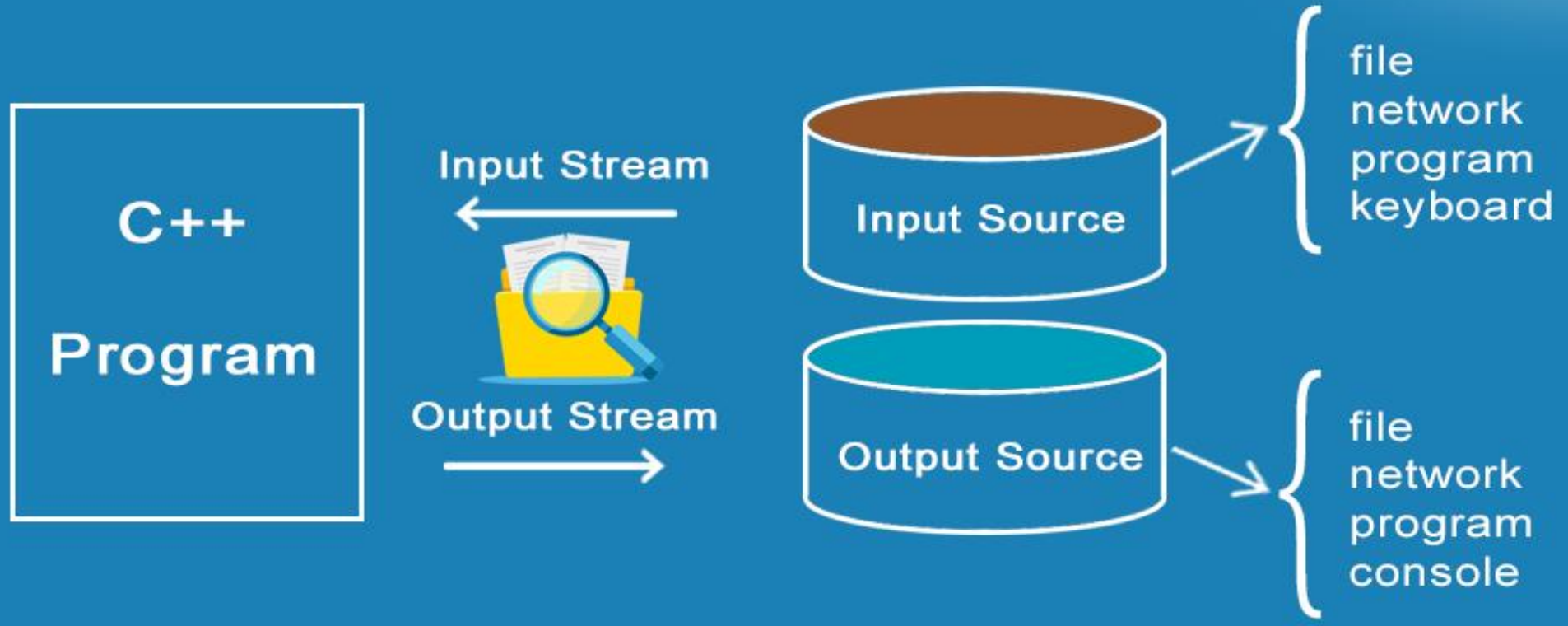
OUTPUT

```
How are you?
How are y
ou?
```

By – Mohammad Imran

File Handling in C++

MYCPLUS | com



By – Mohammad Imran

File

A collection of data or information that has a name, called the file. Almost all information stored in a computer must be in a file.

Files store data permanently in a storage device. With file handling, the output from a program can be stored in a file. Various operations can be performed on the data while in the file.

By – Mohammad Imran

File

Why Need

When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.

If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C++.

By – Mohammad Imran

File Types

There are two types of file in C++ language.

- ✓ Text Files
- ✓ Binary Files

By – Mohammad Imran

File

Text Files

Text files are the normal **.txt** files. You can easily create text files using simple text editors such as **Notepad**. When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents. They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

By – Mohammad Imran

File

Binary Files

Binary files are mostly the **.bin** files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold a higher amount of data, are not readable easily, and provides better security than text files.

By – Mohammad Imran

File Operation

In C++, you can perform four major operations on files, either text or binary:

- ✓ Creating a new file
- ✓ Opening an existing file
- ✓ Closing a file
- ✓ Reading from and writing information to a file
- ✓ Deleting the files

By – Mohammad Imran

File Operation

C++ provides a comprehensive set of classes and methods in its standard library, mainly the **<fstream>** header, to manage and manipulate files. This enables operations on text files (text mode) and binary files (binary mode).

By – Mohammad Imran

File Operation

Primary Classes

The primary classes for file handling in C++ are:

- ✓ **ofstream:** Represents the output file stream and is used to create and write to files.
- ✓ **ifstream:** Represents the input file stream and is used to read from files.
- ✓ **fstream:** Represents the file stream and can be used for both reading from and writing to files.

File Operation

Primary Functions

C++ provides us with four different operations for file handling. They are:

- ✓ **open()** – This is used to create a file.
- ✓ **write() –** This is used to write new data to file.
- ✓ **read() –** This is used to read the data from the file.
- ✓ **close() –** This is used to close the file.

By – Mohammad Imran

Creating File

Open() Function

The **`open()`** function of **`fstream`** of **`ofstream`** class is used to create a file.

The **`open()`** function is used to read or enter data to a file, we need to open it first. This can be performed with the help of '**`ifstream`**' for reading and '**`fstream`**' or '**`ofstream`**' for writing or appending to the file. All these three objects have **`open()`** function pre-built in them.

Syntax: **`open(FileName, Mode)`**

File Mode

There different mode to open a files are:

Mode	Description
iso::in	File opened in reading mode
iso::out	File opened in write mode
iso::app	File opened in append mode
iso::ate	File opened in append mode.
ios::binary	File opened in binary mode
iso::trunc	File opened in truncate mode
iso::nocreate	The file opens only if it exists
iso::noreplace	The file opens only if it doesn’t exist

By – Mohammad Imran

Creating File

Example - 1

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

int main()

{

   ofstream outfile("Data.txt");

   cout << "File Created";

   return 0;

}
```

OUTPUT

```
File Created
```

NOTE: The location of file will be current directory or bin folder.

By – Mohammad Imran

Writing to File

Example - 2

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ofstream outfile("Data.txt");
   outFile << "Hi, How are you?";
   cout << "File Created";
   cout << "Writing Completed";
   return 0;
}
```

OUTPUT

```
File Created
```

By – Mohammad Imran

Writing to File

Example - 3

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   int roll;
   string name;
   ofstream outFile("D:\\Data.txt");
   cout << "Enter roll: ";
   cin >> roll;
   cin.ignore();
   cout << "Enter name: ";
   getline(cin,name);
   outFile << roll << "," << name;
   cout << "File Created" << endl;
   cout << "Writing Completed";
   return 0;
}
```

By – Mohammad Imran

Reading from File

Example - 4

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   int roll;
   string name;
   ifstream readFile("D:\\Data.txt");
   readFile >> roll;
   readFile >> name;
   cout << roll << name << endl;
   cout << "Data Reading Completed";
   return 0;
}
```

By – Mohammad Imran

Reading Data to Existing File

open() Function

In order to read the information from the file, we need first to open it. The opening of the file is done using **`ofstream`** or **`fstream`** object of the file. Files in C++ can be opened in different modes depending on the purpose of writing or reading. Hence, we need to specify the mode of file opening along with the file name.

By – Mohammad Imran

Reading Data to Existing File

open() Function

There are basically 3 default modes that are used while opening a file in C++:

- ✓ **ofstream ios :: out – To Write Data**
- ✓ **fstream ios :: in | ios :: out – To Read and Write Data**
- ✓ **ofstream ios :: in – To Read Data**

Syntax:

```
void open(filename, ios :: openmodemode_name);
```

Reading Data to Existing File

open() Function

We can simply read the information from the file using the operator **(>>)** with the file's name. We need to use the fstream or ifstream object in C++ to read the file. Reading of the file line by line can be done by simply using the while loop along with the function of **`ifstream 'getline()'`**.

Closing the File:

As we all know about C++ memory management, as the program terminates, it frees all the allocated memory and the used resources. However, it is generally considered good practice to close the file after the desired operations are performed.

By – Mohammad Imran

Reading Data to Existing File

Example - 5

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ifstream infile;
   infile.open("D:\\Data.txt", ios::in);
   if (!infile)
   {
      cerr << "File Not Fount" << endl;
      return 1;
   }
   while (!infile.eof())  //while (infile >> noskipws >> ch) {
   {                      //    cout << ch;
      char ch;
      infile >> noskipws >> ch;
      cout << ch;
   }
   return 0;
}
```

By – Mohammad Imran

Append Data to Existing File

We can append data to existing file with previous record by using file mode **`ios::app`** with existing file name. To do this we need to open file using **`open()`** function.

Append Data to Existing File

Example - 6

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ofstream outfile;
   outfile.open("D:\\Data.txt", ios::app);
   string str;
   cout << "Enter data : ";
   getline(cin,str);
   outfile << str << endl;
   outfile.close();
   cout << "Data writing successful." << endl;
   return 0;
}
```

By – Mohammad Imran

Writing and Reading Data

Example - 7

```
#include <iostream>
#include <fstream>
#include <string>
class FileHandler
{
  private:
   string filename;
  public:
   FileHandler(const string& fname) : filename(fname)
   {}
   void writeData()
   {
      ofstream outfile(filename);
      if (!outfile)
      {
```

By – Mohammad Imran

Writing and Reading Data

Example - 7

```
        cerr << "Error opening file for writing!" << endl;
        return;
    }
    string data;
    cout << "Enter data to write (end with 'END'):\n";
    while (true)
    {
        getline(cin, data);
        if (data == "END")
        break; // Stop writing when "END" is entered
        outfile << data << endl;
    }
    outfile.close();
    cout << "Data written to " << filename << endl;
}
```

By – Mohammad Imran

Writing and Reading Data

Example - 7

```
void readData()
{
   ifstream infile(filename);
   if (!infile)
   {
      cerr << "File Not Found!" << endl;
      return;
   }
   string line;
   cout << "Contents of the file:\n";
   while(getline(infile, line))
   {
      cout << line << endl;
   }
   infile.close();
```

By – Mohammad Imran

Writing and Reading Data

Example - 7

```
    }
};

int main()
{
    string fname;
    cout << "Enter file name : ";
    cin >> fname;
    FileHandler fh(fname);
    fh.writeData();
    fh.readData();
    return 0;
}
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

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

struct Student
{
   int id;
   char name[30];
   float grade;
};
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

```
void writeRecords(const string& filename)
{
    ofstream outFile(filename, ios::binary | ios::app);
    if (!outFile)
    {
        cerr << "Error opening file for writing!" << endl;
        return;
    }
    Student student;
    cout << "Enter student ID: ";
    cin >> student.id;
    cout << "Enter student name: ";
    cin.ignore(); // Clear the newline left in the buffer
    cin.getline(student.name, 30);
    cout << "Enter student grade: ";
    cin >> student.grade;
    outFile.write(reinterpret_cast<const char*>(&student), sizeof(Student));
    outFile.close();
    cout << "Record written to file." << endl;
}
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

```
void readRecord(const string& filename, int index)
{
   ifstream inFile(filename, ios::binary);
   if (!inFile)
   {
      cerr << "Error opening file for reading!" << endl;
      return;
   }
   Student student;
   inFile.seekg(index * sizeof(Student), ios::beg);
   inFile.read(reinterpret_cast<char*>(&student),
   sizeof(Student));
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

```
   if (inFile)
   {
      cout << "Record #" << index << ": " << endl;
      cout << "ID: " << student.id << endl;
      cout << "Name: " << student.name << endl;
      cout << "Grade: " << fixed << setprecision(2) <<
      student.grade << endl;
   }
   else
   {
      cerr << "Error reading record #" << index << endl;
   }
   inFile.close();
}
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

```
void displayMenu()
{
   cout << "\nMenu:\n";
   cout << "1. Write Student Record\n";
   cout << "2. Read Student Record\n";
   cout << "3. Exit\n";
   cout << "Choose an option: ";
}

int main()
{
   const string filename = "students.dat";
   int choice;
   do
   {
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

```
displayMenu();
cin >> choice;
switch (choice)
{
   case 1:
      writeRecords(filename);
      break;
   case 2:
      int index;
      cout << "Enter record index to read: ";
      cin >> index;
      readRecord(filename, index);
      break;
```

By – Mohammad Imran

Writing and Reading Data Randomly

Example - 8

```
            case 3:
                cout << "Exiting the program." << endl;
                break;
            default:
                cout << "Invalid choice! Please try again." << endl;
        }
    } while (choice != 3);
    return 0;
}
```

By – Mohammad Imran

Error Handling During File Operation

File Open Check:

- Checks if the file was opened successfully with is_open(). If not, an error message is displayed.

Write Operation Error Check:

- After writing to the file, the code checks if the operation was successful. If writing fails, an error message is shown.

By – Mohammad Imran

Error Handling During File Operation

Seek Operation Check:

- After attempting to move the file pointer with **`seekg()`**, the program checks for failure using **`fail()`**. This ensures that the index is valid before reading.

Read Operation Error Check:

- After reading from the file, it checks whether the read was successful. If not, an error message is displayed.

By – Mohammad Imran

Error Handling During File Operation

Example - 9

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

struct Student
{
   int id;
   char name[30];
   float grade;
};
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
void writeRecords(const std::string& filename)
{
   ofstream outFile(filename, ios::binary | ios::app);
   if (!outFile.is_open())
   {
      cerr << "Error: Unable to open file for writing!" << endl;
      return;
   }
   Student student;
   cout << "Enter student ID: ";
   cin >> student.id;
   cout << "Enter student name: ";
   cin.ignore(); // Clear the newline left in the buffer
   cin.getline(student.name, 30);
   cout << "Enter student grade: ";
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
    cin >> student.grade;
    outFile.write(reinterpret_cast<const char*>(&student),
    sizeof(Student));
    if (!outFile)
    {
       cerr << "Error: Failed to write record to file!" << endl;
    }
    else
    {
       cout << "Record written to file." << endl;
    }
    outFile.close();
}
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
void readRecord(const string& filename, int index)
{
   ifstream inFile(filename, ios::binary);
   if (!inFile.is_open())
   {
      cerr << "Error: Unable to open file for reading!" << endl;
      return;
   }
   Student student;
   inFile.seekg(index * sizeof(Student), ios::beg);
   if (inFile.fail())
   {
      cerr << "Error: Seek failed for record index " << index <<
      "!" << endl;
      inFile.close();
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
        return;
    }
    inFile.read(reinterpret_cast<char*>(&student), sizeof(Student));
    if (!inFile)
    {
        cerr << "Error: Failed to read record #" << index << "!" << endl;
    }
    else
    {
        cout << "Record #" << index << ": " << endl;
        cout << "ID: " << student.id << endl;
        cout << "Name: " << student.name << std::endl;
        cout << "Grade: " << fixed << setprecision(2) << student.grade <<
        endl;
    }
    inFile.close();
}
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
void displayMenu()
{
   cout << "\nMenu:\n";
   cout << "1. Write Student Record\n";
   cout << "2. Read Student Record\n";
   cout << "3. Exit\n";
   cout << "Choose an option: ";
}
int main()
{
   const string filename = "students.dat";
   int choice;
   do
   {
      displayMenu();
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
cin >> choice;
switch (choice)
{
   case 1:
      writeRecords(filename);
      break;
   case 2:
      int index;
      cout << "Enter record index to read: ";
      cin >> index;
      readRecord(filename, index);
      break;
```

By – Mohammad Imran

Error Handling During File Operation

Example - 9

```
        case 3:
            cout << "Exiting the program." << std::endl;
            break;
        default:
            cout << "Invalid choice! Please try again." << endl;
    }
  }while (choice != 3);
  return 0;
}
```

By – Mohammad Imran

Coding Questions

By – Mohammad Imran

Question - 1

Write a C++ operational program for file handling according to the given menu.

```
---------MENU----------
1. Create File
2. Write Data
3. Read Data
4. Append Data
5. Delete Data
6. Search Data
7. Update Data
0. Exit
-----------------------
Enter Your Choice :
```

Click here to see code

By – Mohammad Imran

QUIZ

By – Mohammad Imran

Quiz - 1

```
#include <iostream>                    //If file Data.txt not
#include <fstream>                     //found then output is
using namespace std;
int main()
{
   ofstream outfile;
   outfile.open("Data.txt",ios::app);
   if(!outfile)
   {
      cout << "File Not Found" << endl;
      return -1;
   }
   outfile << "Hello World" << endl;
   return 0;
}
```

OUTPUT

Data will inserted in the file

By – Mohammad Imran