LAB: 12

OBJECTIVE:

To study the stream classes and File Handling in C++.

Theory:

File handling in C++ refers to the process of creating, reading, writing, and manipulating files

on a storage device, such as a hard drive. This is a crucial aspect of programming, as it allows

data to be stored persistently, even after a program has ended. C++ provides robust file han-

dling capabilities through the <fstream> library, which includes classes designed specifically

for file operations.

Key Concepts

1. File Streams:

o Stream: A stream is an abstraction that represents a sequence of bytes. In the

context of file handling, streams are used to perform input and output operations

on files.

o File Stream Classes: The <fstream> library in C++ provides three main classes

to handle files:

▪ ifstream (Input File Stream): Used to read data from files.

▪ ofstream (Output File Stream): Used to write data to files.

▪ fstream (File Stream): Used for both reading from and writing to files.

2. File Modes:

o File Modes: When opening a file, you can specify the mode in which the file

should be opened. Common file modes include:

▪ std::ios::in: Open for reading.

▪ std::ios::out: Open for writing.

▪ std::ios::app: Open for appending (writing at the end of the file).

▪ std::ios::binary: Open in binary mode.

▪ std::ios::trunc: If the file exists, its content is deleted before opening.

3. File Operations:

o Opening a File: Files can be opened using the open() function or directly in the

constructor of the file stream object. It's important to check if the file was suc-

cessfully opened before proceeding with operations.

o Reading from a File: Data can be read using methods like getline() for reading

line by line, or by using the stream extraction operator (>>).

o Writing to a File: Data can be written using the stream insertion operator (<<),

similar to how data is output to the console.

o Closing a File: After all operations are done, the file should be closed using the

close() function to free up resources.

4. Error Handling:

o It's important to handle errors, such as the failure to open a file.

a. Program to demonstrate the stream operators (insertion and extraction) overloading.

Source Code:

#include <iostream>

#include <string>

using namespace std;

class Person {

private:

string name;

int age;

public:

Person(string n = "", int a = 0) : name(n), age(a) {}

friend ostream& operator<<(ostream& out, const Person& p);

friend istream& operator>>(istream& in, Person& p);

};

ostream& operator<<(ostream& out, const Person& p) {

out << "Name: " << p.name << ", Age: " << p.age;

return out;

}

istream& operator>>(istream& in, Person& p) {

cout << "Enter name: ";

in >> p.name;

cout << "Enter age: ";

in >> p.age;

return in;

}

int main() {

Person person;

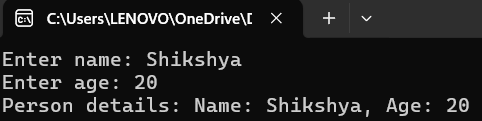
cin >> person;

cout << "Person details: " << person << endl;

return 0;

}

OUTPUT



b. Program to enter the names of any 5 person and store in a text file named “person.txt”.

Source Code:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outfile("person.txt");

if (!outfile) {

cout << "Error opening file!" << endl;

return 1;

}

cout << "Enter 5 names:" << endl;

for (int i = 0; i < 5; ++i) {

string name;

cout << "Name " << (i + 1) << ": ";

cin >> name;

outfile << name << endl;

}

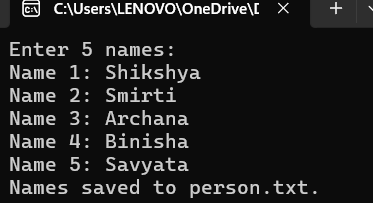
outfile.close();

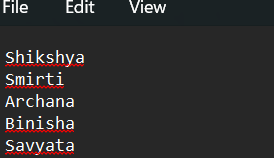
cout << "Names saved to person.txt." << endl;

return 0;

}

OUTPUT:





c. Program to display the content of “person.txt” file on the console.

Source Code:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream infile("person.txt");

if (!infile) {

cout << "Error opening file!" << endl;

return 1;

}

string line;

cout << "Contents of person.txt:" << endl;

while (getline(infile, line)) {

cout << line << endl;

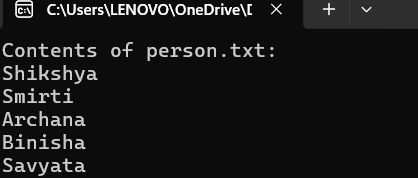
}

infile.close();

return 0;

}

OUTPUT:



d. Program to add more new records in the file “person.txt”.

Source Code:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outfile("person.txt", ios::app);

if (!outfile) {

cout << "Error opening file!" << endl;

return 1;

}

cout << "Enter 3 new names to add to the file:" << endl;

for (int i = 0; i < 3; ++i) {

string name;

cout << "Name " << (i + 1) << ": ";

cin >> name;

outfile << name << endl;

}

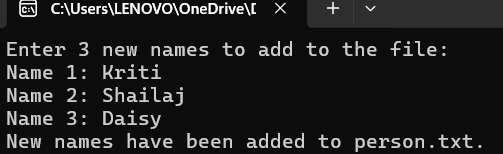
outfile.close();

cout << "New names have been added to person.txt." << endl;

return 0;

}

OUTPUT:





e. Program to copy the contents of one text file to another.

Source Code:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream infile("person.txt");

ofstream outfile("personcopy.txt");

string name;

while (getline(infile, name)) {

outfile << name << endl;

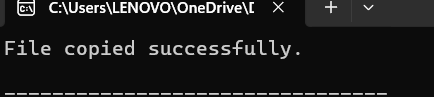
}

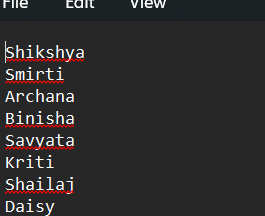
cout << "File copied successfully." << endl;

return 0;

}

OUTPUT:





f. Program that searches for a specific word in a text file and counts the number of times it

appears. The program should be case insensitive.

Source Code:

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

int main() {

ifstream infile("person.txt");

string word, searchWord;

int count = 0;

cout << "Enter word to search: ";

cin >> searchWord;

transform(searchWord.begin(), searchWord.end(), searchWord.begin(), ::tolower);

while (infile >> word) {

transform(word.begin(), word.end(), word.begin(), ::tolower);

if (word == searchWord) count++;

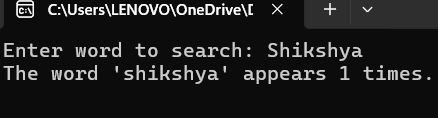
}

cout << "The word '" << searchWord << "' appears " << count << " times." << endl;

return 0;

}

OUTPUT:



g. Program to encrypt the content of a file using a simple algorithm and write the encryptedcontent to another file. Then, write a function to decrypt the file and display the original con-tent.

Source Code:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ifstream infile("person.txt");

ofstream outfile("encrypted.txt");

char ch;

int shift = 3;

while (infile.get(ch))

{

outfile.put(ch + shift);

}

infile.close();

outfile.close();

ifstream encryptedFile("encrypted.txt");

cout << "Decrypted content: ";

while (encryptedFile.get(ch)) {

cout.put(ch - shift);

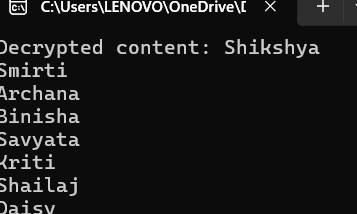
}

encryptedFile.close();

return 0;

}

OUTPUT:





Experiment No:13

Development of a Simple Console Based app using C++.

1. Objective:

To learn the basics about the Development of a Simple Console Based app using C++.

2. Theory:

A console application is a simple program that runs in a command-line interface or console

window, such as Command Prompt in Windows or a terminal in Linux and macOS. Console

applications are text-based programs that interact with users through the command line, with-

out a graphical user interface (GUI). They are often used for small utilities, learning program-

ming concepts, or for performing straightforward tasks.

Key Concepts in Console Applications:

1. Input/Output (I/O):

o Input: In console applications, user input is typically handled using functions

like cin in C++. This allows the user to enter data from the keyboard.

o Output: The output is displayed on the console using cout, which prints text and

data to the console window.

2. Standard Library:

o C++ provides a standard library that includes various functions and objects,

such as iostream, which supports console I/O operations.

o The iostream library contains the cin and cout objects used for input and output,

respectively.

3. Basic Structure:

Every C++ program, including console applications, follows a basic structure that in-

cludes the following components:

Preprocessor Directives: These are instructions given to the compiler before the actual

compilation starts. For example, #include <iostream> tells the compiler to include the

standard input/output stream library needed for input and output operations.

Namespace Declaration: using namespace std; is often used to avoid prefixing std::

before standard functions like cout and cin. The std namespace contains all the standard

C++ library functions and objects.

Main Function: The main() function is the entry point of every C++ console applica-

tion. It's where the execution of the program begins. The return type of main() is usually

int, and it typically returns 0 to indicate successful execution.

Console applications in C++ are foundational to understanding programming logic and structure.

Demonstration:

#include <iostream>

#include <string>

using namespace std;

void displayDietPlan(int day) {

if (day >= 1 && day <= 2) {

cout << "Iron-rich foods: spinach, beans, red meat\n";

} else if (day >= 3 && day <= 4) {

cout << "Protein-rich foods: eggs, fish, chicken\n";

} else if (day >= 5 && day <= 6) {

cout << "Antioxidant-rich foods: berries, dark chocolate\n";

} else if (day >= 7 && day <= 8) {

cout << "- Magnesium-rich foods: bananas, leafy greens\n";

} else if (day >= 9 && day <= 10) {

cout << "Healthy fats: avocado, nuts, seeds\n";

} else {

cout <<"Invalid day entered.Please enter a day between 1 and 10.\n";

}

}

int main() {

int day;

string name;

cout << "Welcome to Shikshya Comforting Zone \n";

cout<< "I am here to help you during your menstruation cycle.\n";

cout << "Please enter your name: ";

getline(cin, name);

cout << "Hi "<< name << "! Let's find the best foods to soothe you today.\n";

cout << "Please enter the day of your cycle (1-10): ";

cin >> day;

displayDietPlan(day);

cout << "Remember to listen to your body and take it easy, " << name <<

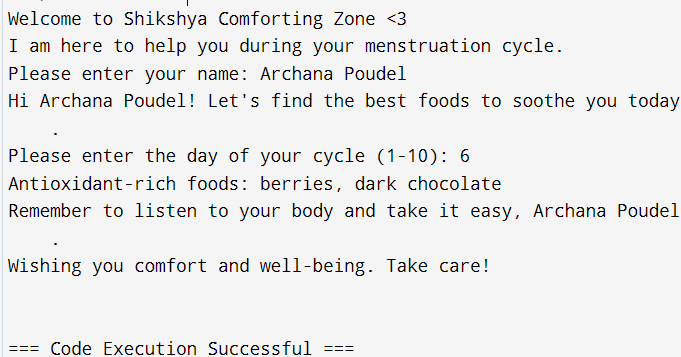
".\n";

cout << "Wishing you comfort and well-being. Take care!\n";

return 0;

}

OUTPUT:

When correct date is entered:

When incorrect date is entered:

