UNIT 5 OOP CODE ASSINMENT

50. Program to Perform Basic Input and Output Using Streams (cin and cout)

```
cpp
CopyEdit
#include <iostream>
using namespace std;

int main() {
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    cout << "You entered: " << number << endl;
    return 0;
}
```

This program demonstrates basic input and output using `cin` to read an integer and `cout` to display the result.

---

51. Program that Reads and Displays Multiple Lines of Text Using cin and cout

```
cpp
CopyEdit
#include <iostream>
#include <string>
using namespace std;

int main() {
    string line;
    cout << "Enter multiple lines of text (Ctrl+Z to end input):" << endl;

    while (getline(cin, line)) {
        cout << "You entered: " << line << endl;
    }

    return 0;
}
```

This program reads multiple lines of text using `getline()` and displays each line until the user stops input (Ctrl+Z in Windows, Ctrl+D in Linux/Mac).

---

52. Program that Uses Streams to Read Integers from the User and Display Their Sum

```
cpp
CopyEdit
```

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

int main() {
    int num1, num2;
    cout << "Enter two integers: ";
    cin >> num1 >> num2;
    cout << "The sum is: " << num1 + num2 << endl;
    return 0;
}
```

This program reads two integers from the user and calculates their sum using `cin` and `cout`.

---

53. Program to Input and Output Various Data Types Using cin and cout

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

int main() {
    int age;
    double salary;
    string name;

    cout << "Enter your name: ";
    cin >> name;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Enter your salary: ";
    cin >> salary;

    cout << "\nName: " << name << endl;
    cout << "Age: " << age << endl;
    cout << "Salary: " << fixed << setprecision(2) << salary << endl;

    return 0;
}
```

This program demonstrates how to input and output different data types: string, integer, and floating point, using `cin` and `cout`.

---

54. Program that Formats Output Using Manipulators such as setw, setprecision, and fixed

```
cpp
CopyEdit
```

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

int main() {
    double pi = 3.14159265359;

    cout << "Formatted output:" << endl;
    cout << "Pi to 2 decimal places: " << fixed << setprecision(2) << pi <<
endl;
    cout << "Pi to 5 decimal places: " << fixed << setprecision(5) << pi <<
endl;
    cout << "Right-aligned number with width 10: " << setw(10) << 1234 <<
endl;
    cout << "Left-aligned number with width 10: " << setw(10) << left << 1234
<< endl;

    return 0;
}
```

55. Program that Reads User Input for Name, Age, and Salary, and Then Displays the Information Using Formatted Output

```
cpp
CopyEdit
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    string name;
    int age;
    double salary;

    // Taking input
    cout << "Enter your name: ";
    cin >> name;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Enter your salary: ";
    cin >> salary;

    // Displaying formatted output
    cout << "\n---- Employee Information ----" << endl;
    cout << "Name   : " << name << endl;
    cout << "Age    : " << age << endl;
    cout << "Salary : " << fixed << setprecision(2) << salary << endl;

    return 0;
}
```

This program reads the user's name, age, and salary, and then displays the information with formatted output for the salary.

56. Program to Demonstrate the Use of ifstream and ofstream for File Input and Output

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

int main() {
    // Writing to a file
    ofstream outFile("example.txt");
    if (outFile.is_open()) {
        outFile << "Hello, this is a test file!" << endl;
        outFile << "We are writing to this file using ofstream.";
        outFile.close();
        cout << "Data written to file successfully!" << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    // Reading from a file
    string line;
    ifstream inFile("example.txt");
    if (inFile.is_open()) {
        cout << "\nReading from file:" << endl;
        while (getline(inFile, line)) {
            cout << line << endl;
        }
        inFile.close();
    } else {
        cout << "Unable to open file for reading." << endl;
    }

    return 0;
}
```

This program demonstrates how to use `ifstream` for reading from a file and `ofstream` for writing to a file.

57. Program that Reads a List of Integers from a File and Displays Them on the Console

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

```
int main() {
    int number;
    ifstream inputFile("numbers.txt");

    if (inputFile.is_open()) {
        cout << "Numbers read from file:" << endl;
        while (inputFile >> number) {
            cout << number << endl;
        }
        inputFile.close();
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program reads integers from a file (`numbers.txt`) and displays them on the console.

---

58. Program that Writes a List of Strings to a File

```
cpp
CopyEdit
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
    vector<string> lines = {"First line", "Second line", "Third line"};
    ofstream outputFile("strings.txt");

    if (outputFile.is_open()) {
        for (const string& line : lines) {
            outputFile << line << endl;
        }
        outputFile.close();
        cout << "Strings written to file successfully!" << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    return 0;
}
```

This program demonstrates how to write a list of strings to a file (`strings.txt`).

---

59. Program to Demonstrate Unformatted Input and Output Using get and put Functions

```
cpp
CopyEdit
#include <iostream>
using namespace std;

int main() {
    char ch;
    cout << "Enter a character: ";
    ch = cin.get();  // Unformatted input using get

    cout << "You entered: ";
    cout.put(ch);  // Unformatted output using put
    cout << endl;

    return 0;
}
```

60. Program that Reads and Writes Characters Using get and put

```
cpp
CopyEdit
#include <iostream>
using namespace std;

int main() {
    char character;
    cout << "Enter a character: ";
    character = cin.get();  // Using get to read a single character

    cout << "You entered: ";
    cout.put(character);  // Using put to display the character
    cout << endl;

    return 0;
}
```

This program uses `get()` to read a single character and `put()` to output it. Both are unformatted functions for character input and output.

---

61. Program that Uses Formatted Input and Output to Display a Table of Data

```
cpp
CopyEdit
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    // Setting up table headers
    cout << left << setw(10) << "Name" << setw(10) << "Age" << setw(10) <<
"Salary" << endl;
    cout << "----------------------------" << endl;
    // Input and formatted output for multiple rows
    string name;
    int age;
    double salary;

    for (int i = 0; i < 3; ++i) {
        cout << "Enter name, age, and salary for person " << i+1 << ": ";
        cin >> name >> age >> salary;

        cout << left << setw(10) << name << setw(10) << age << setw(10) <<
fixed << setprecision(2) << salary << endl;
    }

    return 0;
}
```

This program demonstrates formatted input and output using `setw()` for column widths, `left` for left-aligned output, and `setprecision()` for controlling decimal places.

---

62. Program that Uses getline to Read a Full Line of Text and Display It

```
cpp
CopyEdit
#include <iostream>
#include <string>
using namespace std;

int main() {
    string line;
    cout << "Enter a line of text: ";
    getline(cin, line);  // Using getline to read an entire line of text
    cout << "You entered: " << line << endl;
    return 0;
}
```

This program demonstrates the use of `getline()` to read an entire line of text, including spaces, and then displays the line.

---

63. Program that Uses Manipulators to Format Floating-Point Numbers with Different Precisions

```
cpp
CopyEdit
#include <iostream>
#include <iomanip>
using namespace std;
```

```
int main() {
    double pi = 3.14159265359;

    cout << "Pi with different precisions:" << endl;
    cout << "Default precision: " << pi << endl;
    cout << "Precision 2: " << fixed << setprecision(2) << pi << endl;
    cout << "Precision 4: " << fixed << setprecision(4) << pi << endl;
    cout << "Precision 6: " << fixed << setprecision(6) << pi << endl;

    return 0;
}
```

This program uses the `setprecision()` manipulator to control the number of decimal places shown for a floating-point number.

---

64. Program that Uses setw to Align Text Output in Columns

```
cpp
CopyEdit
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    string name = "John";
    int age = 30;
    double salary = 50000.75;

    // Using setw to align text in columns
    cout << left << setw(15) << "Name" << setw(10) << "Age" << setw(15) << 
"Salary" << endl;
    cout << "----------------------------------------" << endl;

    cout << left << setw(15) << name << setw(10) << age << setw(15) << fixed 
<< setprecision(2) << salary << endl;

    return 0;
}
```

65. Program that Uses Manipulators to Format Currency and Percentage Values

```
cpp
CopyEdit
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    double amount = 12345.6789;
    double percentage = 0.185;

    // Formatting currency and percentage
    cout << "Formatted currency: " << fixed << setprecision(2) << "$" <<
amount << endl;
    cout << "Formatted percentage: " << fixed << setprecision(2) <<
percentage * 100 << "%" << endl;

    return 0;
}
```

This program demonstrates how to use manipulators to format currency and percentage values with `fixed` and `setprecision()`.

---

66. Program to Read Data from a Text File and Display It on the Console

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

int main() {
    string line;
    ifstream inputFile("data.txt");

    if (inputFile.is_open()) {
        cout << "Contents of the file:" << endl;
        while (getline(inputFile, line)) {
            cout << line << endl;
        }
        inputFile.close();
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program reads data from a file (`data.txt`) and displays the contents on the console. The file is read line by line using `getline()`.

---

67. Implement a Program to Write User Input to a Text File

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

int main() {
    ofstream outputFile("output.txt");
    if (outputFile.is_open()) {
        string name;
        int age;

        cout << "Enter your name: ";
        cin >> name;
        cout << "Enter your age: ";
        cin >> age;

        outputFile << "Name: " << name << endl;
        outputFile << "Age: " << age << endl;

        outputFile.close();
        cout << "Data written to file successfully!" << endl;
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program writes user input (name and age) to a file (`output.txt`) using `ofstream`.

---

68. Program that Copies the Contents of One File to Another Using File Streams

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

int main() {
    ifstream sourceFile("source.txt", ios::binary);
    ofstream destFile("destination.txt", ios::binary);

    if (sourceFile.is_open() && destFile.is_open()) {
        char ch;
        while (sourceFile.get(ch)) {
            destFile.put(ch);  // Copy each character
        }
        cout << "File copied successfully!" << endl;

        sourceFile.close();
        destFile.close();
    } else {
        cout << "Unable to open files." << endl;
    }

    return 0;
}
```

This program demonstrates how to copy the contents of one file (`source.txt`) to another (`destination.txt`) using file streams.

---

69. Program that Appends New Data to an Existing File

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

int main() {
    ofstream outputFile("data.txt", ios::app);

    if (outputFile.is_open()) {
        string newData;
        cout << "Enter data to append to the file: ";
        cin.ignore();  // To clear the input buffer
        getline(cin, newData);  // To read the entire line

        outputFile << newData << endl;  // Append new data to file
        outputFile.close();

        cout << "Data appended to file successfully!" << endl;
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

70. Program to Read Binary Data from a File Using ifstream

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

int main() {
    ifstream inputFile("binaryfile.dat", ios::binary);

    if (inputFile.is_open()) {
        char ch;
        while (inputFile.get(ch)) {  // Read binary data
            cout.put(ch);  // Display binary data
        }
        inputFile.close();
    } else {
        cout << "Unable to open the file." << endl;
    }
    return 0;
}
```

This program reads binary data from a file (`binaryfile.dat`) and outputs it to the console. It uses `ifstream` with `ios::binary` mode to read binary data.

---

71. Program to Write Binary Data to a File Using ofstream

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

int main() {
    ofstream outputFile("binaryfile.dat", ios::binary);

    if (outputFile.is_open()) {
        char data[] = "This is binary data!";
        outputFile.write(data, sizeof(data) - 1);  // Write binary data
        outputFile.close();
        cout << "Binary data written successfully!" << endl;
    } else {
        cout << "Unable to open the file." << endl;
    }

    return 0;
}
```

This program writes binary data to a file (`binaryfile.dat`) using `ofstream` with `ios::binary` mode.

---

72. Program that Demonstrates the Use of fstream for Both Input and Output Operations

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

int main() {
    fstream file("file.txt", ios::in | ios::out | ios::trunc);

    if (file.is_open()) {
        string data = "This is a test file!";
        file << data << endl;  // Write to file
        file.seekg(0);  // Move to the beginning of the file for reading
        string line;
        cout << "Reading from file: " << endl;
        while (getline(file, line)) {  // Read from file
            cout << line << endl;
        }
        file.close();
    } else {
        cout << "Unable to open the file." << endl;
    }

    return 0;
}
```

This program uses `fstream` to perform both input and output operations. It writes to the file and then reads from the same file.

---

73. Program to Read and Write Complex Data Structures to a File Using Binary File Streams

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

struct Employee {
    int id;
    char name[50];
};

int main() {
    Employee emp1 = {101, "John Doe"};

    ofstream outFile("employee.dat", ios::binary);
    if (outFile.is_open()) {
        outFile.write(reinterpret_cast<char*>(&emp1), sizeof(emp1));  //
Write structure
        outFile.close();
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    Employee emp2;
    ifstream inFile("employee.dat", ios::binary);
    if (inFile.is_open()) {
        inFile.read(reinterpret_cast<char*>(&emp2), sizeof(emp2));  // Read
structure
        inFile.close();
        cout << "Employee ID: " << emp2.id << ", Name: " << emp2.name <<
endl;
    } else {
        cout << "Unable to open file for reading." << endl;
    }

    return 0;
}
```

This program demonstrates how to read and write a structure (in this case, `Employee`) to a binary file using binary file streams.

---

74. Program to Rename and Delete Files Using the rename and remove Functions

```
cpp
CopyEdit
#include <iostream>
#include <cstdio>  // For rename() and remove()
using namespace std;

int main() {
    const char* oldName = "oldfile.txt";
    const char* newName = "newfile.txt";

    if (rename(oldName, newName) == 0) {
        cout << "File renamed successfully!" << endl;
    } else {
        cout << "Unable to rename file." << endl;
    }

    if (remove(newName) == 0) {
        cout << "File deleted successfully!" << endl;
    } else {
        cout << "Unable to delete file." << endl;
    }

    return 0;
}
```

75. Implement a Program to Create, Open, and Close Files Using File Streams

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

int main() {
    ofstream outputFile("newfile.txt");  // Create and open file for writing

    if (outputFile.is_open()) {
        outputFile << "This is a new file." << endl;
        cout << "File created and written successfully!" << endl;
        outputFile.close();  // Close the file
    } else {
        cout << "Unable to create or open the file." << endl;
    }

    ifstream inputFile("newfile.txt");  // Open file for reading
    if (inputFile.is_open()) {
        string line;
        while (getline(inputFile, line)) {
            cout << line << endl;
        }
        inputFile.close();  // Close the file
    } else {
        cout << "Unable to open file for reading." << endl;
    }

    return 0;
}
```

This program creates a new file (`newfile.txt`), writes to it, and then reads from it. It also demonstrates opening and closing files using `ofstream` and `ifstream`.

---

76. Create a Program that Uses the seekg and tellg Functions to Manipulate File Pointers

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

int main() {
    ifstream inputFile("data.txt", ios::in);

    if (inputFile.is_open()) {
        // Move the file pointer to the 5th byte
        inputFile.seekg(5, ios::beg);
        cout << "File pointer position after seekg: " << inputFile.tellg() <<
endl;

        string data;
        inputFile >> data;  // Read data after moving the pointer
        cout << "Data read from the file: " << data << endl;

        inputFile.close();
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program uses `seekg()` to move the file pointer and `tellg()` to get the current position of the pointer in the file.

---

77. Write a Program that Uses the seekp and tellp Functions to Set and Retrieve the Put Pointer Position

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

int main() {
    ofstream outputFile("output.txt", ios::out);

    if (outputFile.is_open()) {
        // Move the put pointer to the 10th byte
        outputFile.seekp(10, ios::beg);
        cout << "Put pointer position after seekp: " << outputFile.tellp() <<
endl;

        outputFile << "Hello, this is a test!";
        outputFile.close();
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program demonstrates the use of `seekp()` to move the put pointer and `tellp()` to get its current position during output operations.

---

78. Write a Program to Open a File in Different Modes (Read, Write, Append) and Demonstrate Their Effects

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

int main() {
    // Open file in read mode
    ifstream inputFile("data.txt");
    if (inputFile.is_open()) {
        cout << "File opened in read mode." << endl;
        string line;
        while (getline(inputFile, line)) {
            cout << line << endl;
        }
        inputFile.close();
    } else {
        cout << "Unable to open file in read mode." << endl;
    }

    // Open file in write mode
    ofstream outputFile("output.txt");
    if (outputFile.is_open()) {
        outputFile << "Writing new data to file." << endl;
        outputFile.close();
    } else {
        cout << "Unable to open file in write mode." << endl;
    }

    // Open file in append mode
    ofstream appendFile("output.txt", ios::app);
    if (appendFile.is_open()) {
        appendFile << "Appending new data to the file." << endl;
        appendFile.close();
    } else {
        cout << "Unable to open file in append mode." << endl;
    }

    return 0;
}
```

This program demonstrates the use of different file modes: read, write, and append. It opens a file in each mode and performs respective operations.

---

79. Implement a Program that Reads from and Writes to a File in Binary Mode

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

struct Data {
    int id;
    char name[50];
};

int main() {
    Data data1 = {101, "John Doe"};

    // Write binary data to file
    ofstream outputFile("binaryfile.dat", ios::binary);
    if (outputFile.is_open()) {
        outputFile.write(reinterpret_cast<char*>(&data1), sizeof(data1));
        outputFile.close();
        cout << "Binary data written successfully!" << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    // Read binary data from file
    Data data2;
    ifstream inputFile("binaryfile.dat", ios::binary);
    if (inputFile.is_open()) {
        inputFile.read(reinterpret_cast<char*>(&data2), sizeof(data2));
        inputFile.close();
        cout << "Data read from binary file: " << endl;
        cout << "ID: " << data2.id << ", Name: " << data2.name << endl;
    } else {
        cout << "Unable to open file for reading." << endl;
    }

    return 0;
}
```

80. Create a Program that Demonstrates the Difference Between Text and Binary File Modes

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

int main() {
    // Writing in text mode
    ofstream textFile("textfile.txt");
    if (textFile.is_open()) {
        textFile << "This is a text file." << endl;
        textFile.close();
        cout << "Text file written successfully." << endl;
    } else {
        cout << "Unable to open text file." << endl;
    }

    // Writing in binary mode
    ofstream binaryFile("binaryfile.dat", ios::binary);
    if (binaryFile.is_open()) {
        int number = 123;
        binaryFile.write(reinterpret_cast<char*>(&number), sizeof(number));
        binaryFile.close();
        cout << "Binary file written successfully." << endl;
    } else {
        cout << "Unable to open binary file." << endl;
    }

    return 0;
}
```

This program demonstrates the difference between text and binary file modes. The text file stores data as characters, while the binary file stores the data as raw binary values.

---

81. Write a Program to Open a File in Truncation Mode and Demonstrate Its Effect

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

int main() {
    // Open file in truncation mode (default mode for ofstream)
    ofstream outputFile("truncationfile.txt", ios::trunc);
    if (outputFile.is_open()) {
        outputFile << "This file will be overwritten." << endl;
        outputFile.close();
        cout << "File written using truncation mode." << endl;
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program demonstrates the truncation mode by opening a file and writing data into it. If the file already exists, it will be overwritten.

---

82. Write a Program to Read and Write Binary Data to a File Using the Read and Write Functions

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

struct Data {
    int id;
    char name[50];
};

int main() {
    Data data1 = {100, "Jane Doe"};

    // Write binary data
    ofstream binaryOutFile("datafile.dat", ios::binary);
    if (binaryOutFile.is_open()) {
        binaryOutFile.write(reinterpret_cast<char*>(&data1), sizeof(data1));
        binaryOutFile.close();
        cout << "Data written to binary file." << endl;
    } else {
        cout << "Unable to open binary file for writing." << endl;
    }

    // Read binary data
    Data data2;
    ifstream binaryInFile("datafile.dat", ios::binary);
    if (binaryInFile.is_open()) {
        binaryInFile.read(reinterpret_cast<char*>(&data2), sizeof(data2));
        binaryInFile.close();
        cout << "Data read from binary file: " << endl;
        cout << "ID: " << data2.id << ", Name: " << data2.name << endl;
    } else {
        cout << "Unable to open binary file for reading." << endl;
    }

    return 0;
}
```

This program writes and reads data in binary format using the `write` and `read` functions with `ofstream` and `ifstream`.

---

83. Implement a Program That Uses Random Access to Read and Write Data at Specific Positions in a Binary File

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

struct Data {
    int id;
    char name[50];
};

int main() {
    Data data1 = {200, "Alice Smith"};

    // Write data to a binary file
    ofstream binaryOutFile("randomaccess.dat", ios::binary);
    if (binaryOutFile.is_open()) {
        binaryOutFile.write(reinterpret_cast<char*>(&data1), sizeof(data1));
        binaryOutFile.close();
        cout << "Data written to binary file at position 0." << endl;
    } else {
        cout << "Unable to open binary file for writing." << endl;
    }
    // Read data from a specific position in the binary file
    ifstream binaryInFile("randomaccess.dat", ios::binary);
    if (binaryInFile.is_open()) {
        binaryInFile.seekg(0, ios::beg);  // Move to the beginning
        Data data2;
        binaryInFile.read(reinterpret_cast<char*>(&data2), sizeof(data2));
        cout << "Data read from binary file: " << endl;
        cout << "ID: " << data2.id << ", Name: " << data2.name << endl;
        binaryInFile.close();
    } else {
        cout << "Unable to open binary file for reading." << endl;
    }

    return 0;
}
```

This program demonstrates random access by reading and writing data at specific positions in a binary file using `seekg()` for reading.

---

84. Create a Program That Reads and Writes a Structure to a Binary File Using Random Access

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

struct Person {
    int id;
    char name[50];
};

int main() {
    Person person1 = {1, "John Doe"};

    // Write structure to binary file at position 0
    ofstream outFile("structure.dat", ios::binary);
    if (outFile.is_open()) {
        outFile.seekp(0, ios::beg);  // Seek to position 0
        outFile.write(reinterpret_cast<char*>(&person1), sizeof(person1));
        outFile.close();
        cout << "Structure written to binary file." << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    // Read structure from binary file
    Person person2;
    ifstream inFile("structure.dat", ios::binary);
    if (inFile.is_open()) {
        inFile.seekg(0, ios::beg);  // Seek to position 0
        inFile.read(reinterpret_cast<char*>(&person2), sizeof(person2));
        inFile.close();
        cout << "Data read from binary file: " << endl;
        cout << "ID: " << person2.id << ", Name: " << person2.name << endl;
    } else {
        cout << "Unable to open file for reading." << endl;
    }

    return 0;
}
```

85. Write a Program That Updates Specific Records in a Binary File Using Random Access

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

struct Student {
    int id;
    char name[50];
};

int main() {
    Student student1 = {1, "Alice"};
    Student student2 = {2, "Bob"};

    // Write data to binary file
    ofstream outFile("students.dat", ios::binary);
    if (outFile.is_open()) {
        outFile.write(reinterpret_cast<char*>(&student1), sizeof(student1));
        outFile.write(reinterpret_cast<char*>(&student2), sizeof(student2));
        outFile.close();
        cout << "Data written to binary file." << endl;
    } else {
        cout << "Unable to open binary file for writing." << endl;
    }

    // Update record at position 1 (second record)
    fstream file("students.dat", ios::binary | ios::in | ios::out);
    if (file.is_open()) {
        Student updatedStudent = {2, "Robert"}; // Update Bob's name to
Robert
        file.seekp(sizeof(Student), ios::beg);  // Move to the second record
        file.write(reinterpret_cast<char*>(&updatedStudent),
sizeof(updatedStudent));
        file.close();
        cout << "Record updated successfully." << endl;
    } else {
        cout << "Unable to open file for updating." << endl;
    }
    return 0;
}
```

This program demonstrates how to update a specific record in a binary file using random access with `seekp()`.

---

86. Implement a Program That Reads and Displays the Contents of a Binary File in Reverse Order

```
cpp
CopyEdit
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

struct Data {
    int id;
    char name[50];
};

int main() {
    // Write some data to the binary file
    ofstream outFile("reverse.dat", ios::binary);
    if (outFile.is_open()) {
        Data data1 = {1, "Alice"};
        Data data2 = {2, "Bob"};
        Data data3 = {3, "Charlie"};

        outFile.write(reinterpret_cast<char*>(&data1), sizeof(data1));
        outFile.write(reinterpret_cast<char*>(&data2), sizeof(data2));
        outFile.write(reinterpret_cast<char*>(&data3), sizeof(data3));
        outFile.close();
        cout << "Data written to binary file." << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    // Read and display data in reverse order
    ifstream inFile("reverse.dat", ios::binary);
    if (inFile.is_open()) {
        vector<Data> dataVec;
        Data temp;
        while (inFile.read(reinterpret_cast<char*>(&temp), sizeof(temp))) {
            dataVec.push_back(temp);
        }
        inFile.close();

        // Display in reverse order
        cout << "Data in reverse order:" << endl;
        for (int i = dataVec.size() - 1; i >= 0; i--) {
            cout << "ID: " << dataVec[i].id << ", Name: " << dataVec[i].name
<< endl;
        }
    } else {
        cout << "Unable to open file for reading." << endl;
    }

    return 0;
}
```

This program reads data from a binary file into a `vector` and then displays the contents in reverse order.

---

87. Write a Program That Uses Streams to Read User Input, Process It, and Write the Results to a File

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

int main() {
    int num1, num2;
    cout << "Enter two integers: ";
    cin >> num1 >> num2;

    // Process the numbers (sum)
    int sum = num1 + num2;

    // Write the result to a file
    ofstream outFile("result.txt");
    if (outFile.is_open()) {
        outFile << "The sum of " << num1 << " and " << num2 << " is: " << sum
<< endl;
        outFile.close();
        cout << "Result written to file." << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }

    return 0;
}
```

This program reads user input, processes the input (calculates the sum), and writes the result to a file.

---

88. Implement a Program That Reads a Configuration File and Uses Its Settings to Control Program Behavior

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

int main() {
    string line;
    ifstream configFile("config.txt");

    if (configFile.is_open()) {
        while (getline(configFile, line)) {
            if (line == "enable_feature_x") {
                cout << "Feature X is enabled." << endl;
            } else if (line == "enable_feature_y") {
                cout << "Feature Y is enabled." << endl;
            } else {
                cout << "Unknown setting: " << line << endl;
            }
        }
        configFile.close();
    } else {
        cout << "Unable to open configuration file." << endl;
    }

    return 0;
}
```

This program reads a configuration file and uses its settings to control the program's behavior by checking specific lines in the file.

---

89. Create a Program That Logs Error Messages to a File Using File Streams

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

void logError(const string& errorMessage) {
    ofstream logFile("error_log.txt", ios::app); // Open in append mode
    if (logFile.is_open()) {
        logFile << "Error: " << errorMessage << endl;
        logFile.close();
        cout << "Error logged successfully." << endl;
    } else {
        cout << "Unable to open log file." << endl;
    }
}

int main() {
    string errorMessage = "An unexpected error occurred.";
    logError(errorMessage);

    return 0;
}
```

90. Write a Program That Uses File Streams to Create a Simple Text Editor

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

int main() {
    string text;
    cout << "Enter text to save to the file (type 'EXIT' to stop):" << endl;
    ofstream outFile("text_editor.txt");

    if (outFile.is_open()) {
        while (true) {
            getline(cin, text);
            if (text == "EXIT") {
                break;
            }
            outFile << text << endl;
        }
        outFile.close();
        cout << "Text saved to text_editor.txt." << endl;
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program simulates a simple text editor where the user can type lines of text and save them to a file. The program stops when the user types `"EXIT"`.

---

91. Implement a Program That Reads and Processes a CSV File Using File Streams

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

```
#include <sstream>
#include <string>
using namespace std;

int main() {
    ifstream inFile("data.csv");
    string line, word;

    if (inFile.is_open()) {
        while (getline(inFile, line)) {
            stringstream ss(line);
            while (getline(ss, word, ',')) {
                cout << word << " ";
            }
            cout << endl;
        }
        inFile.close();
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program reads data from a CSV file and processes each line by splitting the words separated by commas, displaying them on the console.

---

92. Create a Program That Uses File Streams to Search for a Specific Word in a Text File and Count Its Occurrences

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

int main() {
    ifstream inFile("text.txt");
    string word, line;
    int count = 0;

    if (inFile.is_open()) {
        cout << "Enter the word to search for: ";
        cin >> word;

        while (getline(inFile, line)) {
            size_t pos = line.find(word);
            while (pos != string::npos) {
                count++;
                pos = line.find(word, pos + 1);
            }
        }
        inFile.close();
        cout << "The word '" << word << "' occurred " << count << " times."
<< endl;
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

This program searches for a specific word in a text file and counts how many times it occurs.

---

93. Write a Program That Demonstrates the Use of Exception Handling with File Operations

```
cpp
CopyEdit
#include <iostream>
#include <fstream>
#include <exception>
using namespace std;

int main() {
    try {
        ifstream inFile("non_existent_file.txt");

        if (!inFile.is_open()) {
            throw runtime_error("Error: Unable to open file.");
        }

        string content;
        while (getline(inFile, content)) {
            cout << content << endl;
        }
        inFile.close();
    } catch (const runtime_error& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

This program demonstrates exception handling when trying to open a non-existent file. If the file cannot be opened, it throws a `runtime_error` with an appropriate message.

---

94. Implement a Program That Compresses and Decompresses Text Files Using Simple Encoding Techniques

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

// Simple character-based compression by shifting ASCII values
void compressFile(const string& inputFile, const string& outputFile) {
    ifstream inFile(inputFile);
    ofstream outFile(outputFile);
    char ch;

    if (inFile.is_open() && outFile.is_open()) {
        while (inFile.get(ch)) {
            outFile.put(ch + 1);  // Simple encoding by shifting ASCII value
        }
        inFile.close();
        outFile.close();
        cout << "File compressed." << endl;
    } else {
        cout << "Unable to open file." << endl;
    }
}

void decompressFile(const string& inputFile, const string& outputFile) {
    ifstream inFile(inputFile);
    ofstream outFile(outputFile);
    char ch;

    if (inFile.is_open() && outFile.is_open()) {
        while (inFile.get(ch)) {
            outFile.put(ch - 1);  // Decoding by shifting back ASCII value
        }
        inFile.close();
        outFile.close();
        cout << "File decompressed." << endl;
    } else {
        cout << "Unable to open file." << endl;
    }
}

int main() {
    string inputFile = "original.txt";
    string compressedFile = "compressed.txt";
    string decompressedFile = "decompressed.txt";

    compressFile(inputFile, compressedFile);
    decompressFile(compressedFile, decompressedFile);

    return 0;
}
```

95. Create a Program That Uses File Streams to Merge the Contents of Multiple Text Files into a Single File

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

int main() {
    ifstream inFile1("file1.txt");
    ifstream inFile2("file2.txt");
    ofstream outFile("merged.txt");

    if (inFile1.is_open() && inFile2.is_open() && outFile.is_open()) {
        string line;

        // Copy content from file1.txt
        while (getline(inFile1, line)) {
            outFile << line << endl;
        }

        // Copy content from file2.txt
        while (getline(inFile2, line)) {
            outFile << line << endl;
        }

        inFile1.close();
        inFile2.close();
        outFile.close();

        cout << "Files merged successfully into merged.txt." << endl;
    } else {
        cout << "Unable to open one or more files." << endl;
    }

    return 0;
}
```

This program merges the contents of two text files (`file1.txt` and `file2.txt`) and writes them into a new file called `merged.txt`.

---

96. Write a Program That Reads and Processes Large Data Files Using Memory-Mapped Files

Memory-mapped files are used to map a file's contents directly into the address space of the program. It’s a powerful tool for large files. This solution would be platform-dependent, but here’s an example using **Windows** with `mmap`:

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

```
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    const char *filename = "largefile.txt";
    int fd = open(filename, O_RDONLY);
    if (fd == -1) {
        std::cerr << "Error opening file." << std::endl;
        return 1;
    }

    // Get the size of the file
    off_t fileSize = lseek(fd, 0, SEEK_END);

    // Map the file into memory
    char* mappedData = (char*)mmap(0, fileSize, PROT_READ, MAP_PRIVATE, fd,
0);
    if (mappedData == MAP_FAILED) {
        std::cerr << "Error mapping file to memory." << std::endl;
        close(fd);
        return 1;
    }

    // Process file data (simple example: print content)
    std::cout.write(mappedData, fileSize);

    // Clean up
    munmap(mappedData, fileSize);
    close(fd);

    return 0;
}
```

This program maps a file into memory and processes its content. It uses `mmap` for memory mapping, which is common in Unix-based systems (Linux/Unix). This is useful for large files, but note that it's platform-specific.

---

97. Implement a Program That Uses Streams to Perform Basic Encryption and Decryption of Text Files

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

void encryptFile(const string& inputFile, const string& outputFile) {
    ifstream inFile(inputFile);
    ofstream outFile(outputFile);
    char ch;
    if (inFile.is_open() && outFile.is_open()) {
        while (inFile.get(ch)) {
            outFile.put(ch + 1);  // Encrypt by shifting ASCII value
        }
        inFile.close();
        outFile.close();
        cout << "File encrypted." << endl;
    } else {
        cout << "Unable to open file." << endl;
    }
}

void decryptFile(const string& inputFile, const string& outputFile) {
    ifstream inFile(inputFile);
    ofstream outFile(outputFile);
    char ch;

    if (inFile.is_open() && outFile.is_open()) {
        while (inFile.get(ch)) {
            outFile.put(ch - 1);  // Decrypt by shifting ASCII value back
        }
        inFile.close();
        outFile.close();
        cout << "File decrypted." << endl;
    } else {
        cout << "Unable to open file." << endl;
    }
}

int main() {
    string inputFile = "original.txt";
    string encryptedFile = "encrypted.txt";
    string decryptedFile = "decrypted.txt";

    encryptFile(inputFile, encryptedFile);
    decryptFile(encryptedFile, decryptedFile);

    return 0;
}
```

98. Write a Program That Uses Polymorphism to Create a Flexible and Extensible GUI Framework

In C++, creating a GUI framework would generally involve many libraries, such as Qt or GTK. Here's a simplified example of polymorphism in a GUI framework. We'll simulate a basic framework with different types of GUI components:

```
cpp
CopyEdit
#include <iostream>
#include <vector>
using namespace std;
```

```cpp
// Base class for all GUI components
class GUIComponent {
public:
    virtual void render() = 0;  // Pure virtual function
};

// Derived class for Button
class Button : public GUIComponent {
public:
    void render() override {
        cout << "Rendering a Button" << endl;
    }
};

// Derived class for Label
class Label : public GUIComponent {
public:
    void render() override {
        cout << "Rendering a Label" << endl;
    }
};

// Derived class for TextField
class TextField : public GUIComponent {
public:
    void render() override {
        cout << "Rendering a TextField" << endl;
    }
};

// GUI Framework class
class GUIFramework {
public:
    void addComponent(GUIComponent* component) {
        components.push_back(component);
    }

    void renderAll() {
        for (auto& component : components) {
            component->render();
        }
    }

private:
    vector<GUIComponent*> components;
};

int main() {
    GUIFramework framework;

    Button button;
    Label label;
    TextField textField;

    framework.addComponent(&button);
    framework.addComponent(&label);
    framework.addComponent(&textField);
    framework.renderAll();  // Render all components

    return 0;
}
```

In this program, polymorphism allows different types of GUI components (Button, Label, TextField) to be added to the same framework. Each component has its own render method, demonstrating extensibility.

---

99. Implement a Program That Demonstrates the Use of Virtual Functions and Templates to Create a Generic and Reusable Algorithm Library

Here is an example where we use virtual functions with templates to create a reusable algorithm library for calculating the area of various shapes:

```
cpp
CopyEdit
#include <iostream>
#include <cmath>
using namespace std;

// Base class for Shape
class Shape {
public:
    virtual double area() const = 0; // Pure virtual function
    virtual ~Shape() = default;  // Virtual destructor
};

// Derived class for Circle
template <typename T>
class Circle : public Shape {
private:
    T radius;
public:
    Circle(T r) : radius(r) {}

    double area() const override {
        return M_PI * radius * radius;
    }
};

// Derived class for Rectangle
template <typename T>
class Rectangle : public Shape {
private:
    T width, height;
public:
    Rectangle(T w, T h) : width(w), height(h) {}

    double area() const override {
        return width * height;
    }
};

// Generic function to display area of a shape
template <typename T>
void displayArea(Shape* shape) {
    cout << "Area: " << shape->area() << endl;
}

int main() {
    Circle<int> circle(5);
    Rectangle<float> rectangle(4.5, 3.2);

    displayArea<int>(&circle);      // Works with Circle
    displayArea<float>(&rectangle); // Works with Rectangle

    return 0;
}
```

This program uses **virtual functions** and **templates** to create a flexible and reusable algorithm library that can calculate the area of different shapes (Circle, Rectangle). We use `template <typename T>` to allow flexibility with different data types.

---

100. Create a Program That Uses Polymorphism, Templates, and Exception Handling to Implement a Comprehensive and Type-Safe Collection Framework

In this program, we implement a **collection framework** using polymorphism and templates while ensuring that operations like adding, removing, and accessing elements are type-safe. We also use **exception handling** for operations like out-of-bounds access:

```
cpp
CopyEdit
#include <iostream>
#include <vector>
#include <stdexcept>
using namespace std;

// Base class for collection elements
template <typename T>
class Collection {
public:
    virtual void add(const T& element) = 0;
    virtual T get(int index) const = 0;
    virtual void remove(int index) = 0;
    virtual ~Collection() = default;
};

// Derived class for Array-based collection
template <typename T>
class ArrayCollection : public Collection<T> {
```

```
private:
    vector<T> data;
public:
    void add(const T& element) override {
        data.push_back(element);
    }

    T get(int index) const override {
        if (index < 0 || index >= data.size()) {
            throw out_of_range("Index out of range");
        }
        return data[index];
    }

    void remove(int index) override {
        if (index < 0 || index >= data.size()) {
            throw out_of_range("Index out of range");
        }
        data.erase(data.begin() + index);
    }
};

int main() {
    try {
        ArrayCollection<int> collection;
        collection.add(10);
        collection.add(20);
        collection.add(30);

        cout << "Element at index 1: " << collection.get(1) << endl;
        collection.remove(1);

        // This will throw an exception
        cout << "Element at index 1 after removal: " << collection.get(1) <<
endl;

    } catch (const out_of_range& e) {
        cout << "Error: " << e.what() << endl;
    }

    return 0;
}
```