```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
// Function to create a text file
void createTextFile(const string& filename) {
  ofstream outfile(filename);
  if (outfile.is_open()) {
    outfile << "This is a sample text file.\n";
    outfile << "You can add more content here.\n";
    cout << "Text file " << filename << " created successfully!" << endl;
  } else {
    cerr << "Error creating file: " << filename << endl;
  }
  outfile.close(); // Close the file even on errors
}
// Function to read from a text file
void readTextFile(const string& filename) {
  ifstream infile(filename);

  if (infile.is_open()) {
    string line;
    while (getline(infile, line)) {
      cout << line << endl;
    }
  } else {
    cerr << "Error opening file: " << filename << endl;
  }

  infile.close(); // Close the file even on errors
}

// Function to write to a binary file
void writeBinaryFile(const string& filename, const char* data, int size) {
  ofstream outfile(filename, ios::binary);

  if (outfile.is_open()) {
    outfile.write(data, size);
    cout << "Binary data written to file " << filename << endl;
  } else {
    cerr << "Error creating binary file: " << filename << endl;
  }

  outfile.close(); // Close the file even on errors
```

```
}

// Function to read from a binary file
void readBinaryFile(const string& filename, int size) {
  char buffer[size];

  ifstream infile(filename, ios::binary);

  if (infile.is_open()) {
    infile.read(buffer, size);
    cout << "Binary data from file " << filename << ":" << endl;
    for (int i = 0; i < size; ++i) {
      cout << hex << static_cast<int>(buffer[i]) << " ";
    }
    cout << endl;
  } else {
    cerr << "Error opening binary file: " << filename << endl;
  }

  infile.close(); // Close the file even on errors
}

int main() {
  string textFilename = "example.txt";
  string binaryFilename = "data.bin";

  // Create a text file
  createTextFile(textFilename);

  // Read from the text file
  readTextFile(textFilename);

  // Sample data for binary file
  char binaryData[] = "This is binary data";

  // Write to a binary file
  writeBinaryFile(binaryFilename, binaryData, sizeof(binaryData));

  // Read from the binary file (adjust size based on written data)
  readBinaryFile(binaryFilename, sizeof(binaryData));

  return 0;
}
```

```
/*
Text Files:

Student Records: Create a program that allows users to enter student information (name, ID, marks) and store them in a text file. The program should allow users to:

Add new student records.
Display all student records from the file.
Search for a specific student by ID and display their details.
Phonebook: Develop a program that functions as a simple phonebook. Users can:
*/
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Student {
    string name;
    string id;
    int marks;
};

void addStudent() {
    ofstream file("students.txt");
    if (!file) {
        cout<<"Unable to open file for writing"<<endl;
        return;
    }

    Student student;
    cout<<"Enter student name: ";
    cin >> student.name;
    cout<<"Enter student ID: ";
    cin >> student.id;
    cout<<"Enter student marks: ";
    cin >> student.marks;

    file<<student.name<<" "<<student.id<<" "<<student.marks<<endl;
    file.close();
}

void displayStudents() {
```

```
    ifstream file("students.txt");
    if (!file) {
        cout<<"Unable to open file for reading"<<endl;
        return;
    }

    Student student;
    while (file >> student.name >> student.id >> student.marks) {
        cout<<"Name: "<<student.name<<", ID: "<<student.id<<", Marks: "<<student.marks<<endl;
    }

    file.close();
}

void searchStudentById() {
    ifstream file("students.txt");
    if (!file) {
        cout<<"Unable to open file for reading"<<endl;
        return;
    }

    string id;
    cout<<"Enter student ID to search: ";
    cin >> id;

    Student student;
    bool found = false;
    while (file >> student.name >> student.id >> student.marks) {
        if (student.id == id) {
            cout<<"Name: "<<student.name<<", ID: "<<student.id<<", Marks:
"<<student.marks<<endl;
            found = true;
            break;
        }
    }

    if (!found) {
        cout<<"Student with ID "<<id<<" not found"<<endl;
    }

    file.close();
}

int main() {
```

```
    int choice;
    do {
        cout<<"1. Add Student"<<endl;
        cout<<"2. Display Students"<<endl;
        cout<<"3. Search Student by ID"<<endl;
        cout<<"4. Exit"<<endl;
        cout<<"Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addStudent();
                break;
            case 2:
                displayStudents();
                break;
            case 3:
                searchStudentById();
                break;
            case 4:
                cout<<"Exiting"<<endl;
                break;
            default:
                cout<<"Invalid choice nhi chlega yha"<<endl;
        }
    } while (choice != 4);

    return 0;
}



/*
File Encryption/Decryption (Optional): Implement a program that encrypts/decrypts a text file
using a simple Caesar cipher or another basic encryption method.
*/
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

void encryptFile(const string& inputFilename, const string& outputFilename, int shift) {
    ifstream inputFile(inputFilename);
```

```
    ofstream outputFile(outputFilename);

    if (!inputFile || !outputFile) {
        cout<<"Unable to open file" << endl;
        return;
    }

    char ch;
    while (inputFile.get(ch)) {
        if (isalpha(ch)) {
            char base = islower(ch) ? 'a' : 'A';
            ch = (ch - base + shift) % 26 + base;
        }
        outputFile.put(ch);
    }

    inputFile.close();
    outputFile.close();
}

void decryptFile(const string& inputFilename, const string& outputFilename, int shift) {
    encryptFile(inputFilename, outputFilename, 26 - shift);
}

int main() {
    string inputFilename = "plain.txt";
    string encryptedFilename = "encrypted.txt";
    string decryptedFilename = "decrypted.txt";
    int shift = 3;

    encryptFile(inputFilename, encryptedFilename, shift);
    cout << "File encrypted successfully." << endl;

    decryptFile(encryptedFilename, decryptedFilename, shift);
    cout << "File decrypted successfully." << endl;

    return 0;
}

/*****************************************************************************
```

Image Copy: Write a program that copies the contents of an image file (e.g., JPG, PNG) to a new file.
Ensure you handle binary data correctly.

```
*******************************************************************************/
#include <fstream>
#include <iostream>

using namespace std;

void copyImage(const string& srcFilename, const string& destFilename) {
    ifstream srcFile(srcFilename, ios::binary);
    ofstream destFile(destFilename, ios::binary);

    if (!srcFile || !destFile) {
        cout<<"Unable to open file" << endl;
        return;
    }

    destFile << srcFile.rdbuf();

    srcFile.close();
    destFile.close();
}

int main() {
    string srcFilename = "source.jpg";
    string destFilename = "copy.jpg";

    copyImage(srcFilename, destFilename);
    cout << "Image copied successfully." << endl;

    return 0;
}

#include <fstream>
#include <iostream>
#include <string>

using namespace std;

struct Item {
    char name[50];
    double price;
    int quantity;
};

void addItem() {
```

```
    ofstream file("inventory.dat", ios::binary | ios::app);
    if (!file) {
        cerr << "Unable to open file for writing" << endl;
        return;
    }

    Item item;
    cout << "Enter item name: ";
    cin.ignore();
    cin.getline(item.name, 50);
    cout << "Enter item price: ";
    cin >> item.price;
    cout << "Enter item quantity: ";
    cin >> item.quantity;

    file.write(reinterpret_cast<char*>(&item), sizeof(Item));
    file.close();
}

void displayItems() {
    ifstream file("inventory.dat", ios::binary);
    if (!file) {
        cerr << "Unable to open file for reading" << endl;
        return;
    }

    Item item;
    while (file.read(reinterpret_cast<char*>(&item), sizeof(Item))) {
        cout << "Name: " << item.name << ", Price: " << item.price << ", Quantity: " <<
item.quantity << endl;
    }

    file.close();
}

void updateItemQuantity() {
    fstream file("inventory.dat", ios::binary | ios::in | ios::out);
    if (!file) {
        cerr << "Unable to open file for reading/writing" << endl;
        return;
    }

    char name[50];
    int newQuantity;
```

```
    cout << "Enter item name to update quantity: ";
    cin.ignore();
    cin.getline(name, 50);
    cout << "Enter new quantity: ";
    cin >> newQuantity;

    Item item;
    bool found = false;
    while (file.read(reinterpret_cast<char*>(&item), sizeof(Item))) {
        if (strcmp(item.name, name) == 0) {
            item.quantity = newQuantity;
            file.seekp(-static_cast<int>(sizeof(Item)), ios::cur);
            file.write(reinterpret_cast<char*>(&item), sizeof(Item));
            found = true;
            break;
        }
    }

    if (!found) {
        cout << "Item with name " << name << " not found" << endl;
    }

    file.close();
}

int main() {
    int choice;
    do {
        cout << "1. Add Item" << endl;
        cout << "2. Display Items" << endl;
        cout << "3. Update Item Quantity" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addItem();
                break;
            case 2:
                displayItems();
                break;
            case 3:
                updateItemQuantity();
```

```
            break;
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 4);

    return 0;
}

#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Contact {
    string name;
    string phoneNumber;
};

void addContact() {
    ofstream file("phonebook.txt", ios::app);
    if (!file) {
        cerr << "Unable to open file for writing" << endl;
        return;
    }

    Contact contact;
    cout << "Enter contact name: ";
    cin >> contact.name;
    cout << "Enter phone number: ";
    cin >> contact.phoneNumber;

    file << contact.name << " " << contact.phoneNumber << endl;
    file.close();
}

void searchContactByName() {
    ifstream file("phonebook.txt");
    if (!file) {
        cerr << "Unable to open file for reading" << endl;
```

```
        return;
    }

    string name;
    cout << "Enter contact name to search: ";
    cin >> name;

    Contact contact;
    bool found = false;
    while (file >> contact.name >> contact.phoneNumber) {
        if (contact.name == name) {
            cout << "Name: " << contact.name << ", Phone Number: " << contact.phoneNumber <<
endl;
            found = true;
            break;
        }
    }

    if (!found) {
        cout << "Contact with name " << name << " not found" << endl;
    }

    file.close();
}

int main() {
    int choice;
    do {
        cout << "1. Add Contact" << endl;
        cout << "2. Search Contact by Name" << endl;
        cout << "3. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addContact();
                break;
            case 2:
                searchContactByName();
                break;
            case 3:
                cout << "Exiting..." << endl;
                break;
```

```
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 3);

    return 0;
}
```

TRY AND CATCH

```
#include <iostream>
using namespace std;
float division(int x, int y) {
    if (y == 0) {
        throw "Attempted to divide by zero!";
    }
    return (x / y);
}
int main() {
    int i = 25;
    int j = 0;
    float k = 0;
    try {
        k = division(i, j);
        cout << k << endl;
    } catch (const char* e) {
        cerr << e << endl;
    }

    return 0;
}]
```

Calculator usingv try and catch

```
#include <iostream>
```

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

float add(float a, float b) {
    return a + b;
}

float subtract(float a, float b) {
    return a - b;
}

float multiply(float a, float b) {
    return a * b;
}

float divide(float a, float b) {
    if (b == 0) {
        throw runtime_error("Division by zero error");
    }
    return a / b;
}

int main() {
    float num1, num2;
    char operation;
    float result;

    cout << "Enter first number: ";
    cin >> num1;

    cout << "Enter an operator (+, -, *, /): ";
    cin >> operation;

    cout << "Enter second number: ";
    cin >> num2;

    try {
        switch (operation) {
            case '+':
                result = add(num1, num2);
                break;
            case '-':
                result = subtract(num1, num2);
                break;
```

```
            case '*':
                result = multiply(num1, num2);
                break;
            case '/':
                result = divide(num1, num2);
                break;
            default:
                throw invalid_argument("Invalid operator");
        }
        cout << "Result: " << result << endl;
    } catch (const runtime_error& e) {
        cerr << e.what() << endl;
    } catch (const invalid_argument& e) {
        cerr << e.what() << endl;
    }

    return 0;
}
```

Q.What are the advantages and disadvantages of using exceptions in C++ compared to traditional error codes?
Ans:

Advantages of Using Exceptions in C++:

1. Separation of Error Handling Code
2. Automatic Resource Management.
3. Error Propagation
4. Consistent Error Handling
5. Enhanced Readability
6. Handling Unrecoverable Errors

Disadvantages of Using Exceptions in C++:

1. Performance Overhead
2. Complexity in Control Flow

Q. How can you ensure that exception classes provide informative error messages for debugging?

To ensure that exception classes provide informative error messages for debugging, you can follow these best practices:

1. Use Meaningful Names
2. Detailed Error Messages

3. Include Relevant Data
4. Exception Hierarchy

Q. Discuss strategies for optimizing exception handling performance, especially in performance-critical applications.

Optimizing exception handling performance in performance-critical applications involves several strategies. Here are some key approaches:

1. Minimize Use of Exceptions

2. Optimize Exception Handling Code

3. Reduce the Cost of Throwing Exceptions

4. Efficient Use of Try-Catch Blocks

5. Compiler Optimizations

6. Use Alternative Error Handling Techniques

7. Platform-Specific Optimizations

Q. How can you design a hierarchy of exception classes for improved code maintainability and reusability?
Ans:

1. Define a Clear Purpose and Scope

2. Base Exception Class

3. Intermediate Exception Classes

4. Specific Exception Classes

5. Exception Handling Strategy

Q. When might it be appropriate to not use exceptions in C++ for error handling?
Ans: reasons where it may be more appropriate to avoid using exceptions for error handling:-

1. Performance-Critical Applications
2. Simple Error Handling Requirements
3. Compatibility with C Libraries
4. Predictability and Control Flow
5. Resource-Constrained Environments

/***

Develop a C++ program that demonstrates robust exception handling for file operations.

The program should:

Read data from a text file.
Validate the data format (e.g., expecting specific number of values per line).
Perform calculations based on the valid data.
Implement exception handling for the following error scenarios:

File opening failure: Throw a custom exception named FileOpenError if the file cannot be opened.
Invalid data format: Throw a custom exception named InvalidDataFormatException if a line in the file doesn't match the expected format.
Calculation errors: Throw a custom exception named CalculationError with a descriptive message if any calculation fails (e.g., division by zero).

***/

```
#include<bits/stdc++.h>

using namespace std;

struct Data {
   int value1;
   int value2;
};

vector<Data> readFile(const string& filename) {
```

```
    ifstream file(filename);
    if (!file.is_open()) {
        throw FileOpenError(filename);
    }

    vector<Data> data;
    string line;
    while (getline(file, line)) {
        istringstream iss(line);
        Data d;
        if (!(iss >> d.value1 >> d.value2)) {
            throw InvalidDataFormatException(line);
        }
        data.push_back(d);
    }

    return data;
}

double performCalculation(int a, int b) {
    if (b == 0) {
        throw CalculationError("Division by zero");
    }
    return static_cast<double>(a) / b;
}

int main() {
    try {
        vector<Data> data = readFile("data.txt");

        for (const auto& d : data) {
            try {
                double result = performCalculation(d.value1, d.value2);
                cout << "Result of " << d.value1 << " / " << d.value2 << " = " << result << endl;
            } catch (const CalculationError& e) {
                cerr << e.what() << endl;
            }
        }
    } catch (const FileOpenError& e) {
        cerr << e.what() << endl;
    } catch (const InvalidDataFormatException& e) {
        cerr << e.what() << endl;
    } catch (const exception& e) {
        cerr << "Unexpected error: " << e.what() << endl;
```

```
    }

    return 0;
}
```