Final Exam Paper Breakdown

Question #1 (Code Writing) [40: 20 each]

Part (a)

- **Topic**: **Hierarchy of Shapes**
    - **Key Concept**: Class hierarchy for shapes (probably involves base and derived classes such as Shape, `Circle`, `Rectangle`, etc.).
    - **Task**: Write a code for a class that reuses data and functions across multiple shape classes.
    - **Key Ideas**: You'll likely need to implement **inheritance** and **polymorphism**. Look at the constructor and member function reuse in derived shapes.

Sample Structure for Part (a):

```
class Shape {
protected:
    string name;
    string type;
    int borderWidth;
    Color borderColor;
    Color fillColor;
public:
    virtual void draw() = 0;  // Pure virtual function
    void setShape(string n, string t, int bw, Color bc, Color fc) {
        name = n;
        type = t;
        borderWidth = bw;
        borderColor = bc;
        fillColor = fc;
    }
    // More functions related to shapes
};

class Rectangle : public Shape {
    float length;
    float width;
public:
    void setRectangle(string n, string t, int bw, Color bc, Color fc, Point2D p,
float l, float w) {
        Shape::setShape(n, t, bw, bc, fc);
        // set position and other members
        length = l;
        width = w;
    }
    void draw() override {
        // Drawing logic for rectangle
    }
};
```

Part (b)

- **Topic**: **Abstract Class with Virtual Functions**
  - **Key Concept**: Writing an abstract base class and overriding virtual functions in child classes.
  - **Task**: Define an abstract class with a pure virtual function `show` and override it in child classes.
  - **Key Ideas**: You'll define **abstract classes** and **virtual functions**, and implement child classes that override the `show` function.

Sample Structure for Part (b):

```
class AbstractClass {
public:
    virtual void show() = 0;  // Pure virtual function
};

class DerivedClass : public AbstractClass {
public:
    void show() override {
        cout << "Derived Class Show Function" << endl;
    }
};
```

Question #2 (Theory + Code Examples) [40: 8 each]

- This section is **theory-based**, requiring you to describe concepts clearly and use small code examples to support your answers.

Topics Covered:

- **Polymorphism**: **Definition**, use of **base class pointers** to call derived class methods.
- **Inheritance Types**: **Multiple inheritance**, **multilevel inheritance**, etc.
- **Virtual Functions**: Explanation of **runtime polymorphism**.

Question #3 (Templates + Exceptions + Multidimensional Arrays + Class Inheritance) [40: 10 each]

Part (a) - Template Class with Multiple Overloaded Operators

You will be asked to create a **template class** that can handle different data types, likely storing them in arrays and overloading the necessary operators.

Part (b) - Exception Handling + Multidimensional Arrays

You will also work with **exception handling** (try-catch) in the context of **2D or 3D arrays** and throw appropriate exceptions.

Sample Code for Template Class + Exception Handling:

```
template <typename T>
class MyArray {
private:
    T* data;
    int size;
public:
    MyArray(int n) : size(n) {
        data = new T[n];
    }
    ~MyArray() {
        delete[] data;
    }
    T get(int index) {
        if (index >= size || index < 0) throw out_of_range("Index out of
bounds");
        return data[index];
    }
    void set(int index, T value) {
        if (index >= size || index < 0) throw out_of_range("Index out of
bounds");
        data[index] = value;
    }
};
```

Question #4 (Various Class Designs) [40: 10 each]

Part (a) - Fixed Length Data Member Class + Binary File Handling

You need to write a class where every data member is of a fixed length, and you'll be interacting with a binary file stream (`fstream`). The task likely involves **random access** and file **manipulation** (seekg, seekp).

Sample Code for Part (a) with Binary File Handling:

```
class FixedLengthData {
private:
    char name[50];
    int value;
public:
    void writeToFile(fstream& fs) {
        fs.seekg(0, ios::beg);
        fs.write(reinterpret_cast<char*>(this), sizeof(*this));
    }
};
```

Part (b) - Text Class Similar to String

Create a class `Text` similar to the built-in `string` class. This will likely involve **dynamic memory management** and **operator overloading**.

Part (c) - Array of Pointers + nullptr Handling

The task will likely involve working with **arrays of pointers** that may contain `nullptr` values, where you'll handle the null pointers properly.

Part (d) - Queue and Stack Classes

Here, you'll be dealing with two important data structures, **Queue** and **Stack**, using **templates** and **pointer-based memory allocation**.

```
template <typename T>
class Queue {
private:
    T* arr;
    int front, rear;
public:
    void enqueue(T val) { /* insert code for enqueue operation */ }
    T dequeue() { /* insert code for dequeue operation */ }
    bool isEmpty() { return front == rear; }
};
```

Key Concepts to Focus on:

- **Polymorphism** (Upcasting/Downcasting)
- **Template classes** and **exceptions**
- **Binary file handling** (random access, writing objects)
- **Queue & Stack** (LIFO, FIFO)
- **Inheritance** (multiple, multilevel)
- **Abstract Classes** (pure virtual functions)

Conclusion:

The **final paper** focuses on the key concepts you've studied in class:

- Object-Oriented Programming (OOP) (Inheritance, Polymorphism, Virtual functions)
- Templates and Exception Handling
- Data Structures (Queue, Stack, Array-based operations)
- Binary File Handling

Let me know if you want me to expand on any specific question with more code or theory for this exam paper!