National University of Computer and Emerging Sciences

Karachi Campus

Karachi Campus

Object-oriented Programming (CS-1004) Final Exam

Date: 19/5/2025

Total Time: 3hrs

Total Marks: **50**

Total Questions: 04

Course Instructor(s)

Basit Ali, Minhal, Sumaiya, Nida, Sobia, Abeeha, Atiya, Abeer, Bakhtawar.

Roll No _____ Section _____ Student Signature _____

Do not write below this

Attempt all questions.

CLO 1: *Identify real world problems in terms of objects rather than procedure* **[Marks 25, 60 min]**

Q1:Part A: Find the errors and provide the corrected version of the codes given.

1.template <class T1,class T2>
class myclass {
  T1 num1;
  T2 num2;
public:
  myclass() {
    cout << "Default constructor" << endl;
  }
  myclass(T1 a, T2 b) {
    num1 = a;
    num2 = b;
  }
  void add() {
    int sum = num1 + num2;
    cout << "The value of sum is " << sum; }
};
int main() {
  myclass obj1;
  obj1.add();
  myclass<int> obj(3, 4);
  obj.add();
  return 0;
}

Solution

```
#include <iostream>
using namespace std;
template <class
  T1,class T2>
class myclass {
  T1 num1;
  T2 num2;
public:
  myclass() {
    cout << "Default
  constructor" <<
  endl;
   num1 = T1();  //
  Initialize to default
  value
    num2 = T1();
}
  myclass(T1 a, T2 b) {
    num1 = a;
    num2 = b;
  }
  void add() {
    int sum = num1 +
  num2;
    cout << "The
  value of sum is " <<
  sum << endl;
  }
};
int main() {
 myclass<int,int> obj1;
  obj1.add();  // Will
  show 0 if both
  default-initialized
myclass<int,int>
  obj(3, 4);
  obj.add();
```

```
2 .class Complex {
    float real;
    float imag;

  public:
    Complex(float r, float i) {
       real = r;
       imag = i;
    }

    Complex operator+(Complex c) {
       Complex temp;
       temp.real = real + c.real;
       temp.imag = imag + c.imag;
       return temp;
    }

    void display() {
       cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complex c1(3.2, 4.5), c2(1.1, 2.2);
    Complex c3 = c1 + c2;
    c3.display();
    return 0;
}
```

```
return 0}
```

Solution

```
#include <iostream>
using namespace std;
class Complex {
   float real;
   float imag;

public:
   Complex(float r,
   float i) {
      real = r;
      imag = i;
   }
Complex() : real(0),
   imag(0) {}
   Complex
   operator+(Complex
   c) {
      Complex temp;
      temp.real = real +
   c.real;
      temp.imag = imag
   + c.imag;
      return temp;
   }
void display() {
      cout << real << " +
   " << imag << "i" <<
   endl;
   }
};

int main() {
   Complex c1(3.2,
   4.5), c2(1.1, 2.2);
   Complex c3 = c1 +
   c2;
   c3.display();
   return 0;
}
```

and also const Complex& c: Improves efficiency by avoiding unnecessary copying.

```
3.class A{
int x;
public: A(int i) { x = i; }
void show() { cout << x; }
};
class B: virtual public A{
public: B(int i):A(i) { }
};
class C: virtual public A{
public: C(int i):B(i) { }
};
class D: public B, public C {
```

Solution

```
#include <iostream>
using namespace std;
class A {
   int x;
public:
   A(int i) { x = i; }
   void show() { cout
   << x; }
};
class B : virtual public
   A {
public:
```

```
};
int main(){
D d(5);
d.show();
return 0;
}
```

```
   B(int i) : A(i) { }
};
class C : virtual public
   A {
public:
   C(int i) : A(i) { }
};
class D : public B,
   public C {
public:
   D(int i) : A(i), B(i),
   C(i) { }
};
int main() {
   D d(5);
   d.show();
   return 0;
}
```

4.
```
#include <iostream>
#include <string>
using namespace std;

class DivisionError {
   string msg;
};

class Calculator {
public:
   static double divide(int a, int b) {
      if (b == 0)
         throw DivisionError("Cannot divide by
   zero");
      if (a < 0 || b < 0)
         throw "Negative input not allowed";
      return static_cast<double>(a) / b;
   }

   static void compute() {
      try {
         cout << "Result: " << divide(10, 2) <<
   endl;
         cout << "Result: " << divide(-5, 1) <<
   endl;  // Will throw: string
         cout << "Result: " << divide(8, 0) << endl;
   // Won't be reached
      }
      catch (const DivisionError& e) {
         e.showMessage();
         try {
            cout << "Retrying...\n";
            divide(6, 0);  // Throws again
         }
         catch (const DivisionError& ex) {
            ex.showMessage();
         }
      }
      catch (const char* msg) {
```

Solution
```
#include <iostream>
#include <string>
using namespace std;
class DivisionError {
   string msg;
public:
   DivisionError(string
   m) : msg(m) {}
   void
   showMessage()
   const {
      cout << "Division
   Error: " << msg <<
   endl;
   }
};
class Calculator {
public:
   static double
   divide(int a, int b) {
      if (b == 0)
         throw
   DivisionError("Cann
   ot divide by zero");
      if (a < 0 || b < 0)
         throw
   "Negative input not
   allowed";
      return
   static_cast<double>
   (a) / b;
   }
 static void compute()
   {
      try {
         cout << "Result:
   " << divide(10, 2) <<
   endl;
         cout << "Result:
   " << divide(-5, 1) <<
   endl;  // Will throw:
   string
```

```
        cout << "Caught string exception: " <<
    msg << endl;
        try {
            cout << "Trying a safe division...\n";
            cout << "Result: " << divide(8, 2) <<
    endl;  // Safe call
        }
        catch (...) {
            cout << "Unexpected exception during
    safe retry\n";
        }
      }
      catch (...) {
        cout << "Unknown error occurred!" <<
    endl;
      }
    }
};

int main() {
    Calculator::compute();
    return 0;
}
```

```
cout << "Result: " <<
    divide(8, 0) << endl;
    // Won't be reached
      }
     catch (const
    DivisionError& e) {

    e.showMessage();
          try {
             cout <<
    "Retrying...\n";
             divide(6, 0);
    // Throws again
          }
          catch (const
    DivisionError& ex) {

    ex.showMessage();}
      }
catch (const char*
    msg) {
          cout << "Caught
    string exception: "
    << msg << endl;
     try {
             cout <<
    "Trying a safe
    division...\n";
             cout <<
    "Result: " <<
    divide(8, 2) << endl;
    // Safe call
          }
          catch (...) {
   cout << "Unexpected
    exception during
    safe retry\n"; }   }
       catch (...) {
          cout <<
    "Unknown error
    occurred!";}
    }};

int main() {

    Calculator::compute
    ();
    return 0;
}
```

Q1:Part B: Find the output of the codes given. There are no syntax errors.

1.
```
class SecureVault {
   int secretKey;
   string password;
public:
   SecureVault(int key, string
pwd) : secretKey(key),
password(pwd){}
   friend class Hacker;
   void reveal() const { cout <<
"Key: " << secretKey << " |
Password: " << password <<
endl; }};
class Hacker {
public:
   void crack(SecureVault& vault)
{
       vault.secretKey = 999;
       cout << "Cracked: " <<
 vault.password << endl;}};
int main() {
   SecureVault vault(42,
"Admin@123");
   Hacker evil;
   evil.crack(vault);
   vault.reveal();
   return 0;}
```

Cracked: Admin@123
Key: 999 | Password: Admin@123

2.
```
class Universe
   { static int count;
   int id;
public:
   Universe() : id(++count) { cout <<
"Universe #" << id << " created." <<
endl; }
   ~Universe() { cout << "Universe #"
<< id << " destroyed." << endl; }
   static void reset() { count = 0; }
};
int Universe::count = 0;
int main() {
   Universe::reset();
   Universe u1, u2;
   {
      Universe u3;
   }
   Universe* u4 = new Universe();
   delete u4;
   return 0;
}
```

Universe #1 created.
Universe #2 created.
Universe #3 created.
Universe #3 destroyed.
Universe #4 created.
Universe #4 destroyed.
Universe #2 destroyed.
Universe #1 destroyed.

3.
```
class Core {
public:
   Core() { cout << "Core()" << endl; }
   virtual ~Core() { cout << "~Core()" <<
endl; }
   virtual void execute() = 0;};
class PluginA : virtual public Core
{ public:
   PluginA() { cout << "PluginA()" << endl; }
   void execute() override { cout <<
"PluginA::execute()" << endl; }};
class PluginB : virtual public Core {
public:
   PluginB() { cout << "PluginB()" << endl; }
   void execute() override { cout <<
"PluginB::execute()" << endl; }};
class Hybrid : public PluginA, public PluginB {
public:
   Hybrid() { cout << "Hybrid()" << endl; }
   void execute() override {
      cout << "Hybrid::execute()" << endl;
      PluginA::execute();
      PluginB::execute(); }};
int main() {
   Core* system = new Hybrid();
   system->execute();
   delete system; return 0;}
```

Core()
PluginA()
PluginB()
Hybrid()
Hybrid::execute()
PluginA::execute()
PluginB::execute()
~Core()

National University of Computer and Emerging Sciences

Karachi Campus

Q1:Part C: Complete the missing codes based on the code snippets and outputs provided. Assume there is no mistake in the code provided and required libraries are included.

1.
```
int myFunction(int value){
        if (value == 0)
                return 0;
        else if (value == 10)
                return 1;
        else
                throw 100;
}
int main() {
        int num1 = 0, num2 = 10,
num3 = 100;
        int res1, res2, res3;
        /*_____*/
```

Solution

```
try {

 res1 = myFunction(num1);

 res2 = myFunction(num2);

  res3 = myFunction(num3);

        }

        catch (int i) {

                cout << i << endl;

                cout << res1 << ", " <<
res2 << endl;

        }
```

Output:

100

0, 1

2.
```
/*_____*/
int main() {
        string fileName[2] =        {
"input.txt", "output.txt" };

writeStringToFile(fileName[1],
readStringFromFile(fileName[0]
));


        return 0;
}
```

Solution

```
string readStringFromFile(string
fileName) {
        fstream fin(fileName, ios::in);
        string str;
        getline(fin, str);
        return str;
}
void writeStringToFile(string fileName,
string data) {
        fstream fout(fileName,
ios::out);
        string str = data + " " + data;
        fout << str;
}
```

Output:

Input.txt Contents:

Hello World!!

Output.txt Contents:

Hello World!! Hello World!!

Input.txt Contents:

Bye World!!

Output.txt Contents:

Bye World!! Bye World!!

3.
```
template<typename T>
bool greaterThan(T val1, T val2) {
        if (val1 > val2)
                return true;
        else
                return false;}
/*_____
*/ int main() {
        int i1 = 10, i2 = 20;
        float f1 = 10.1f, f2 = 20.2f;
        Q4 obj1(10), obj2(20);

        cout << greaterThan(i1, i2) << ",
" << greaterThan(i2, i1) << endl;
        cout << greaterThan(f1, f2) <<
", " << greaterThan(f2, f1) << endl;
        cout << greaterThan(obj1, obj2)
<< ", " << greaterThan(obj2, obj1) <<
endl;
        return 0;}
```

Solution

```
class Q4 {
        int q4;

public:
        Q4 (int a) {
                q4 = a;
        }
        bool operator > (Q4& obj) {
                if (this->q4 > obj.q4)
                        return true;
                else
                        return false;
        }
};
```

Output:

0, 1

0, 1

0, 1

CLO 4: *Illustrate Object-Oriented design artifacts and their mapping to Object-Oriented Programming using C++.* **[Marks 10, 45 min]**

Q2: In the heat of the 2025 Indo-Pak digital conflict, Pakistan's elite cyber unit—Unit Zero-Hawk—launches a top-secret cyber offensive, code-named Operation Bunyan-ul-Marsoos. The operation involves breaching India's encrypted cyber defense system and deploying stealth payloads through a C++-driven terminal simulation.

Phase 1: Mission Setup – Abstract Class Design (1 mark)

Define an abstract class CyberOperation containing the following pure virtual functions:

- accessIntel() – Reads data from the binary file, decrypts payloads using Caesar cipher from Phase 3, and displays only undeleted records.
- extractPayload() – Extracts records with high risk level (≥ 8) based on riskLevel logic from Phase 2 and saves them into a separate file as described in Phase 4.
- deleteTraces() – Marks high-risk records as deleted using logic from Phase 5.

Implement these pure virtual functions in a derived class (e.g., Operation).If necessary, declare friend class Operation; in the IntelRecord class to permit file-level data access OR add public getters/setters if needed.

Phase 2: Data Infiltration – Binary File Creation (3 marks)

Create a class IntelRecord with the following attributes: int recordID, char targetLocation[50], int riskLevel, char payload[100], bool isDeleted. Implement the following functions:

- input() – Takes user input for all fields and encrypts the payload using Caesar cipher with a shift of +3.
- display(bool decrypt) – Displays record details; if decrypt is true, decrypts the payload using Caesar cipher (relevant for Phase 3).
- getRiskLevel() – Returns the risk level of the record; used in Phase 4 and Phase 5.
- isMarkedDeleted() – Returns true if the record is already marked as deleted; used in Phase 3.
- markDeleted() – Sets the deletion status to true; used in Phase 5.

Phase 3: Accessing Classified Data (2 marks)

Open intel.dat in binary mode, read all records, and display only those where isDeleted is false. Each displayed record must have its payload decrypted using Caesar cipher with a shift of -3.

Caesar Cipher (Explanation):It shifts each character by a fixed number of positions.

Example: Original text = attack, shift +3 ⟶ dwwdfn To decrypt, shift -3 ⟶ dwwdfn ⟶ attack

Phase 4: High-Risk Extraction (1 mark)

Find all records with riskLevel >= 8 (as defined in Phase 2) and copy them into a file named extracted.dat.

If no such record exists, throw ExtractionException.

Phase 5: Digital Sanitization (1 mark)

Open intel.dat and mark all records with riskLevel >= 8 (refer to Phase 2) as deleted using markDeleted().

If no records are updated, throw DeleteTraceException.

Phase 6: Exception Handling War Room (2 marks) Define the following custom exceptions:

- FileAccessException – Thrown when a file fails to open("File could not be opened!").
- ExtractionException – Thrown when no high-risk data is found during extraction ("No high-risk records found for extraction!")(used in Phase 4).
- DeleteTraceException – Thrown when no records are marked as deleted (used in Phase 5).("No traces marked as deleted!")

Solution

```
#include <iostream>
#include <fstream>

using namespace std;

// Phase 6: Exception Handling War Room
class FileAccessException {
public:
   const char* what() const {
      return "File could not be opened!";
   }
};
```

```
class ExtractionException {
public:
   const char* what() const {
      return "No high-risk records found for extraction!";
   }
};

class DeleteTraceException {
public:
   const char* what() const {
      return "No traces marked as deleted!";
   }
};

// Phase 1: Abstract Class Design
class CyberOperation {
public:
   virtual void accessIntel() = 0;       // Phase 3
   virtual void extractPayload() = 0;    // Phase 4
   virtual void deleteTraces() = 0;      // Phase 5
};

// Phase 2: IntelRecord Class
class IntelRecord {
   int recordID;
   char targetLocation[50];
   int riskLevel;
   char payload[100];
   bool isDeleted;

public:
   IntelRecord() {
      isDeleted = false;
   }

   void input() {
      cout << "Enter Record ID: ";
      cin >> recordID;
      cin.ignore();
      cout << "Enter Target Location: ";
      cin.getline(targetLocation, 50);
      cout << "Enter Risk Level (1-10): ";
      cin >> riskLevel;
      cin.ignore();
      cout << "Enter Payload: ";
      cin.getline(payload, 100);

      // Caesar cipher encryption (shift +3)
      for (int i = 0; payload[i] != '\0'; i++) {
         payload[i] = payload[i] + 3;
      }

      isDeleted = false;
   }
```

```
    void display(bool decrypt = false) {
        cout << "\nRecord ID: " << recordID;
        cout << "\nTarget Location: " << targetLocation;
        cout << "\nRisk Level: " << riskLevel;
        cout << "\nPayload: ";
        if (decrypt) {
            for (int i = 0; payload[i] != '\0'; i++) {
                cout << char(payload[i] - 3);
            }
        } else {
            cout << payload;
        }
        cout << "\nDeleted: " << (isDeleted ? "Yes" : "No") << endl;
    }

    int getRiskLevel() {
        return riskLevel;
    }

    bool isMarkedDeleted() {
        return isDeleted;
    }

    void markDeleted() {
        isDeleted = true;
    }
};

// Full Operation Implementation
class Operation : public CyberOperation {
public:
    void writeIntelToFile() {
        ofstream outFile("intel.dat", ios::binary | ios::app);
        if (!outFile) throw FileAccessException();

        IntelRecord rec;
        rec.input();
        outFile.write((char*)&rec, sizeof(rec));
        outFile.close();
    }

    void accessIntel() {
        ifstream inFile("intel.dat", ios::binary);
        if (!inFile) throw FileAccessException();

        IntelRecord rec;
        cout << "\n--- Accessing Classified Data ---\n";
        while (inFile.read((char*)&rec, sizeof(rec))) {
            if (!rec.isMarkedDeleted()) {
                rec.display(true); // decrypt = true
            }
        }
        inFile.close();
    }
```

```
    void extractPayload() {
        ifstream inFile("intel.dat", ios::binary);
        ofstream outFile("extracted.dat", ios::binary);

        if (!inFile || !outFile) throw FileAccessException();

        IntelRecord rec;
        bool found = false;

        while (inFile.read((char*)&rec, sizeof(rec))) {
            if (rec.getRiskLevel() >= 8) {
                outFile.write((char*)&rec, sizeof(rec));
                found = true;
            }
        }

        inFile.close();
        outFile.close();

        if (!found) throw ExtractionException();
    }

    void deleteTraces() {
        fstream file("intel.dat", ios::binary | ios::in | ios::out);
        if (!file) throw FileAccessException();

        IntelRecord rec;
        bool deleted = false;
        streampos pos;

        while (file.read((char*)&rec, sizeof(rec))) {
            if (rec.getRiskLevel() >= 8 && !rec.isMarkedDeleted()) {
                pos = file.tellg();
                rec.markDeleted();
                file.seekp(pos - sizeof(rec));  // Move back to start of record
                file.write((char*)&rec, sizeof(rec));  // Overwrite updated record
                deleted = true;
            }
        }

        file.close();

        if (!deleted) throw DeleteTraceException();
    }
};

// Main Simulation
int main() {
    Operation op;
    int choice;

    while (true) {
        cout << "\n=== Operation Banayn-ul-Masroor ===";
        cout << "\n1. Add Intel Record";
```

```
        cout << "\n2. Access Classified Data";
        cout << "\n3. Extract High-Risk Payloads";
        cout << "\n4. Sanitize Digital Traces";
        cout << "\n5. Exit";
        cout << "\nEnter choice: ";
        cin >> choice;

        try {
            switch (choice) {
            case 1:
                op.writeIntelToFile();
                break;
            case 2:
                op.accessIntel();
                break;
            case 3:
                op.extractPayload();
                break;
            case 4:
                op.deleteTraces();
                break;
            case 5:
                cout << "\n\"In the end, it wasn't the bullet that mattered—it was the byte.\"";
                cout << "\n— Unit Zero-Hawk, Post-Mission Debrief\n";
                return 0;
            default:
                cout << "Invalid choice!";
            }
        } catch (FileAccessException& e) {
            cout << "Error: " << e.what() << endl;
        } catch (ExtractionException& e) {
            cout << "Error: " << e.what() << endl;
        } catch (DeleteTraceException& e) {
            cout << "Error: " << e.what() << endl;
        }
    }

    return 0;
}
```

CLO 4: *Illustrate Object-Oriented design artifacts and their mapping to Object-Oriented Programming using C++.* **[Marks 5, 30 min]**
Q3: E-Shop is an online retail platform that allows customers to purchase a variety of products from different vendors. You are required to design the system using the object oriented principles learned.

Customer Class: [1.5 marks]

- Attributes: Attributes: id(int), name(string), Array of Shopping cart (shows products in the cart).Create a Parameterized constructor and a destructor.
- searchProducts(Product *p, product name): searches for a specific product in the list of products. If it's available, display the product or else display "not available".

Product Class: [1 mark]

- Attributes: id(int), name(string), description(string), quantity(int), price(double). Create a Parameterized constructor and a destructor.

Shopping Cart Class: [2.5 marks]

- Attributes: Array of Products (stores products that are currently in the cart)
- 1. void addItem(Product& product, int quantity): adds items if the array is not full and product does not exceed or is in negative quantity.
- 2. double getTotal(): calculates the total bill for the products in the cart.

CLO 5: *Synthesize programs using Generic Programming and exception handling* **[Marks 10, 45 min]**

Solution

```
class Product;
// ShoppingCart Class
class ShoppingCart {
private:
   Product* products;
   int capacity;
   int productCount;

public:
   ShoppingCart(int capacity) : capacity(capacity), productCount(0) {
      products = new Product[capacity]; /
   }

   ~ShoppingCart() {
      delete[] products;
   }


   void addItem(const Product& product, int quantity) {
      if (productCount < capacity && quantity > 0 && product.quantity >= quantity) {
         int existingProductIndex = -1;
         for(int i = 0; i < productCount; i++){
            if(products[i].id == product.id){
               existingProductIndex = i;
               break;
            }
         }
0          if(existingProductIndex != -1){
             products[existingProductIndex].quantity += quantity;
         }
         else{
            products[productCount] = product;
            products[productCount].quantity = quantity;
            productCount++;
         }

         cout << quantity << " " << product.name << "(s) added to cart." << endl;
      } else if (productCount >= capacity) {
         cout << "Cart is full. Cannot add more items." << endl;
      } else if (quantity <= 0) {
         cout << "Invalid quantity. Please enter a positive number." << endl;
      }
       else {
         cout << "Insufficient stock of " << product.name << ". Available quantity: " << product.quantity << endl;
      }
   }

   // Function to calculate the total bill for the products in the cart
   double getTotal() const {
      double total = 0.0;
      for (int i = 0; i < productCount; ++i) {
         total += products[i].price * products[i].quantity;
      }
      return total;
```

```
    }

// Product Class
class Product {
public:
    int id;
    string name;
    string description;
    int quantity;
    double price;

    Product(int id, const string& name, const string& description, int quantity, double price)
        : id(id), name(name), description(description), quantity(quantity), price(price) {}

    ~Product() {}
};

// Customer Class
class Customer {
private:
    int id;
    string name;
    ShoppingCart cart; // Object of ShoppingCart class

public:
    Customer(int id, const string& name, int cartCapacity) : id(id), name(name), cart(cartCapacity) {}

    ~Customer() {}
   void searchProducts(Product* products, int numProducts, const string& productName) const {
        bool found = false;
        for (int i = 0; i < numProducts; ++i) {
            if (products[i].name == productName) {
                cout << "Product found:" << endl;
                cout << "ID: " << products[i].id << ", Name: " << products[i].name
                    << ", Description: " << products[i].description << ", Quantity: "
                    << products[i].quantity << ", Price: $" << fixed << setprecision(2) << products[i].price << endl;
                found = tr0ue;
                break; // Exit the loop since the product is found
            }
        }
        if (!found) {
            cout << "Product \"" << productName << "\" not available." << endl;
        }
    }
```

National University of Computer and Emerging Sciences

Karachi Campus

Q4: Implement a generic ImagePixelCalculator class that performs various image processing operations on generic 2D image matrices. These matrices can hold pixel values in types such as Integer, Float, or Double. Each generic image matrix is represented as a 2D array, where each element represents the pixel value of that image at a specific position.

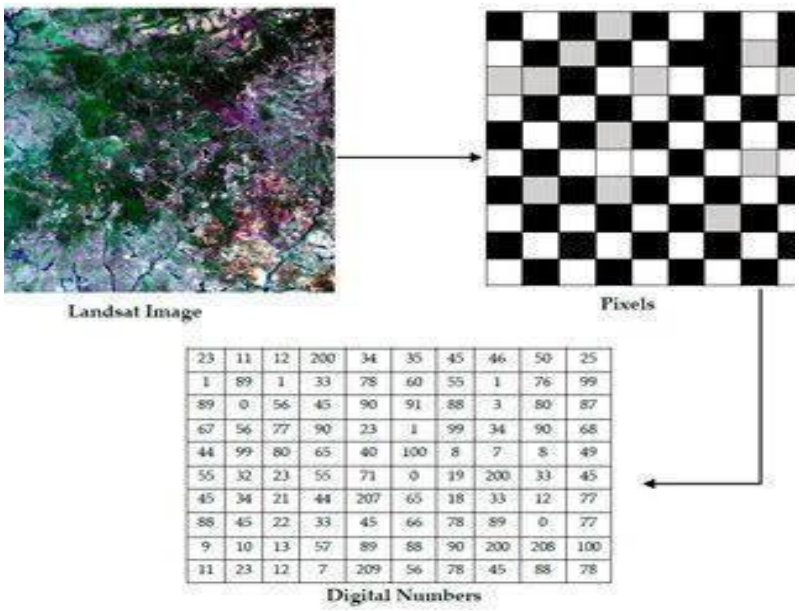
- The generic class should be named ImagePixelCalculator.The class should have two generic attributes, image1 and image2, which store the pixel values of two images. These generic matrices are of any type (supporting Integer, Float, Double, and other numeric types). The generic matrices should be 2D arrays of the same size, meaning both matrices must have the same number of rows and columns**.(1 Mark)**
- The class should have a constructor that initializes the image1 and image2 generic matrices. The constructor should ensure that both generic matrices have the same dimensions (rows and columns). If the dimensions of the matrices do not match, an IllegalArgumentException should be thrown.(**2 Marks)**
- The class should include getter and setter methods for the image1 and image2 attributes to allow access to and modification of the generic 2D matrices. The getter methods should return the generic matrices, and the setter methods should allow the matrices to be set to new values. **(1 Mark)**
- The class should include several generic methods to perform classical image processing operations on the two image matrices: All methods for image processing (denormalizeImage, normalizeImage, applyFilter, smoothImage, transposeImage) have been updated to take two generic 2D image matrices as parameters. The methods perform operations on these matrices of all the types mentioned above.
- **denormalizeImage()**: This method should perform pixel-wise **addition** of corresponding values from image1 and image2 and return the result as a generic 2D matrix of any type . **(1 Mark)**

  normalizeImage(): This method should perform pixel-wise **subtraction** of corresponding values from image1 and image2 and return the result as a 2D matrix of any type**(1 Marks)**

  applyFilter(): This method should perform pixel-wise **multiplication** of corresponding values from image1 and image2 and return the result as a 2D matrix of any type**(1 Mark)**

  smoothImage(): This method should perform pixel-wise **averaging** of corresponding values from image1 and image2 and return the result as a 2D matrix of any type **(1 Mark)**

  transposeImage(): This method should return the transpose of image2 as a 2D matrix. **(1 Mark)**
- **printImageMatrix()**: This method should be a generic method that prints any 2D matrix (of any type to the console. It can be used to display the results of the image processing operations. images in sequence. **(1 Mark)**



solution

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

const int ROWS = 2;
const int COLS = 2;

template<typename T1, typename T2>
class ImagePixelCalculator{
private:
    T1 img1[ROWS][COLS];
    T2 img2[ROWS][COLS];
    double result[ROWS][COLS];

public:
    // Constructor with dimension check
    ImagePixelCalculator(T1 img1Input[ROWS][COLS], T2 img2Input[ROWS][COLS], int rows, int
cols) {
        if (rows != ROWS || cols != COLS) {
            throw invalid_argument("Dimension mismatch: Expected 2x2 matrices.");
        }
        setImage1(img1Input);
        setImage2(img2Input);
    }

    void setImage1(T1 img[ROWS][COLS]) {
        for (int i = 0; i < ROWS; ++i)
            for (int j = 0; j < COLS; ++j)
                img1[i][j] = img[i][j];
    }

    void setImage2(T2 img[ROWS][COLS]) {
        for (int i = 0; i < ROWS; ++i)
            for (int j = 0; j < COLS; ++j)
                img2[i][j] = img[i][j];
    }

    double (*getResult())[COLS] {
        return result;
    }
```

```
void printMatrix(double matrix[ROWS][COLS]) {
    for (int i = 0; i < ROWS; ++i) {
        for (int j = 0; j < COLS; ++j) {
            cout << fixed << setprecision(2) << matrix[i][j] << "\t";
        }
        cout << endl;
    }
    cout << "------------------" << endl;
}

template<typename X = T1, typename Y = T2>
void denormalizeImage() {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            result[i][j] = img1[i][j] + img2[i][j];
}

template<typename X = T1, typename Y = T2>
void normalizeImage() {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            result[i][j] = img1[i][j] - img2[i][j];
}

template<typename X = T1, typename Y = T2>
void applyFilter() {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            result[i][j] = img1[i][j] * img2[i][j];
}

template<typename X = T1, typename Y = T2>
void smoothImage() {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            result[i][j] = (img1[i][j] + img2[i][j]) / 2.0;
}

template<typename T>
void transposeImage(T matrix[ROWS][COLS], T transposed[ROWS][COLS]) {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
```

```
            transposed[j][i] = matrix[i][j];
    }

    void processAndDisplay() {
        cout << "Denormalized Image (Addition):" << endl;
        denormalizeImage();
        printMatrix(result);

        cout << "Normalized Image (Subtraction):" << endl;
        normalizeImage();
        printMatrix(result);

        cout << "Filtered Image (Multiplication):" << endl;
        applyFilter();
        printMatrix(result);

        cout << "Smoothed Image (Averaging):" << endl;
        smoothImage();
        printMatrix(result);

        cout << "Transposed Image2:" << endl;
        double transposed[ROWS][COLS];
        transposeImage(img2, transposed);
        printMatrix(transposed);
    }
};

int main() {
    float img1[ROWS][COLS] = {
        {1.5f, 2.5f},
        {3.5f, 4.5f}
    };

    double img2[ROWS][COLS] = {
        {0.5, 1.0},
        {1.5, 2.0}
    };

    try {
        ImageProcessor<float, double> processor(img1, img2, ROWS, COLS);
        processor.processAndDisplay();
    } catch (const exception& e) {
        cerr << "Error: " << e.what() << endl;
```

```
    }

    return 0;
}
```