Test Questions

// Exam - 1   Time : 3:14 p.m.

// These header directives are used for the entire codes -

```
#include < iostream>

#include <string>

#include <vector>

using namespace std ;
```

Code 1. Polymorphism:

Design a class hierarchy for a simple graphic editor with base class Shape and derived classes Circle, Rectangle, and Triangle. Implement a virtual function draw() in the base class and override it in the derived classes. Write a function that takes a Shape* and calls its draw() method.

```
class Shape {

public:

   virtual void draw() const {

      cout << "Drawing Shape" <<endl;

   } virtual ~Shape() {}

};


class Circle : public Shape {

public:

   void draw() const override {

      cout << "Drawing Circle" <<endl;

   }

};
```

```
class Rectangle : public Shape {
public:
    void draw() const override {
        cout << "Drawing Rectangle" <<endl;
    }
};

class Triangle : public Shape {
public:
    void draw() const override {
        cout << "Drawing Triangle" <<endl;
    }
};

void drawShape(const Shape* shape) {
    shape->draw();
}

int main() {
    Shape* shapes[] = { new Circle(), new Rectangle(), new Triangle() };
    for (Shape* shape : shapes) {
        drawShape(shape);
        delete shape;
    }
    return 0;
}
```

Code 2. Static Members:

Create a class Account that has a static data member totalAccounts to keep track of the number of accounts created. Implement necessary constructors and destructors to update totalAccounts. Write a function to display the total number of accounts.

```
class Account {
private:
    static int totalAccounts;
public:
    Account() {
        totalAccounts++;
    }

    ~Account() {
        totalAccounts--;
    }

    static void displayTotalAccounts() {
        cout << "Total accounts: " << totalAccounts << endl;
    }
};
int Account::totalAccounts = 0;

int main() {
    Account a1, a2;
    Account::displayTotalAccounts();
    {
        Account a3;
        Account::displayTotalAccounts();
    }
    Account::displayTotalAccounts();
    return 0;
}
```

Code 3. Friend Functions:

Implement a class Box that has private data members length, breadth, and height. Write a friend function volume() that calculates and returns the volume of the box. Create objects of Box and use the friend function to compute their volumes.

```
class Box {
private:
    double length, breadth, height;  // private members

public:
    Box(double l, double b, double h) : length(l), breadth(b), height(h) {}

    friend double volume(const Box& b); // friend function declaration
};

    double volume(const Box& b) {         // Friend function defination

       return b.length * b.breadth * b.height;
}

int main() {
    Box b1(4.0, 5.0, 9.0);          // Volume : l*b*h
    cout << "The volume of box is : " << volume(b1) << std::endl;
    return 0;
}
```

Code 4. Templates:

Write a template class Array that can store an array of any data type. Include member functions to perform operations like adding an element, removing an element, and displaying the array. Demonstrate the functionality with different data types.

```
template <typename T>
class Array {
private:
    vector<T> arr;
public:
    void addElement(T element) {
        arr.push_back(element);
    }

    void removeElement() {
        if (!arr.empty()) {
            arr.pop_back();
        } else {
            cout << "Array is empty" << endl;
        }
    }

    void display() const {
        for (const auto& element : arr) {
            cout << element << " ";
        }
        cout << endl;
    }
};

int main() {
    Array<int> intArray;
    intArray.addElement(1);
    intArray.addElement(2);
    intArray.addElement(3);
    intArray.display();
    intArray.removeElement();
    intArray.display();

    Array<string> strArray;
    strArray.addElement("Hello");
    strArray.addElement("World");
    strArray.display();
    strArray.removeElement();
    strArray.display();

    return 0;
}
```

Code 5. Pointers:

Design a class Student with data members name and age. Create an array of Student objects dynamically using pointers. Implement functions to set and display the details of students. Also, write a function to deallocate the memory.

```
class Student {
private:
    string name;
    int age;
public:
    void setDetails(const string& n, int a) {
        name = n;
        age = a;
    }
    void display() const {
        cout << "Student's Name: " << name << ",Student's Age: " << age <<endl;
    }
};

int main() {
    int numStudents;
    cout << "Enter the number of students: ";
    cin >> numStudents;

    Student* students = new Student[numStudents];
    for (int i = 0; i < numStudents; ++i) {
        string name;
        int age;
        cout << "Enter details for student " << i + 1 << " (Name Age): ";
        cin >> name >> age;
        students[i].setDetails(name, age);
    }

    for (int i = 0; i < numStudents; ++i) {
        students[i].display();
    }

    delete[] students;  // deallocation of memory
    return 0;
}
```

6. Polymorphism with Abstract Classes:

Create an abstract class Animal with a pure virtual function sound(). Derive classes Dog, Cat, and Cow from Animal and override the sound() function in each derived class. Write a program to demonstrate polymorphism using these classes.

```
class Animal {                          // Abstract class
public:
    virtual void sound() const = 0;   // pure virtual function
    virtual ~Animal() {}              // destructor
};

class Dog : public Animal {
```

```
public:
    void sound() const override {
        cout << "Dog Barks." <<endl;
    }
};

class Cat : public Animal {
public:
    void sound() const override {
        cout << "Cat Meow." <<endl;
    }
};
class Cow : public Animal {
public:
    void sound() const override {
        cout << "Cow Moo." <<endl;
    }
};

int main() {

    Animal* animals[] = { new Dog(), new Cat(), new Cow() };
    for (Animal* animal : animals) {
        animal->sound();
        // deallocation of memory calls destructor
        delete animal;
    }
    return 0;
}
```

7. Static Member Functions:

Implement a class Math that has static member functions for basic mathematical operations like addition, subtraction, multiplication, and division. Demonstrate the use of these functions without creating an object of the class.

/* **why static used ? Functions are declared as static, they can be called directly using the class name without needing to instantiate an object of the class. */**

```
 class Math {
public:
    static int add(int a, int b) {
        return a + b;
    }
    static int subtract(int a, int b) {
        return a - b;
    }
    static int multiply(int a, int b) {
        return a * b;
    }
};

int main() {
    cout << " Addition of a and b is : " << Math::add(5, 3) <<endl;
    cout << " Subtraction of a and b is : " << Math::subtract(5, 3) <<endl;

    cout << " Multiplication of a and b is : " << Math::multiply(5, 3) <<endl;

    return 0;

}
```

8. Friend Classes:

Create two classes Alpha and Beta. Make Beta a friend class of Alpha so that it can access private data members of Alpha. Implement functions in Beta to manipulate the private data of Alpha.

```
 class Alpha {

private:

    int value;

public:

    Alpha(int v) : value(v) {}

    friend class Beta;

};


class Beta {

public:

    void displayValue(const Alpha& a) {

        cout << "Value: " << a.value << endl;

    }


    void setValue(Alpha& a, int v) {

        a.value = v;

    }
};


int main() {
    Alpha alpha(10);
    Beta beta;


    beta.displayValue(alpha);
    beta.setValue(alpha, 20);
    beta.displayValue(alpha);


    return 0;
}
```

9. Class Templates with Multiple Parameters:

Write a class template Pair that can store a pair of values of any two data types. Include member functions to set and get the values. Demonstrate the usage of this template with different data types.

```
 template <typename T1, typename T2>
class Pair {
private:
    T1 first;
    T2 second;
public:
    void setValues(T1 f, T2 s) {
        first = f;
        second = s;
    }

    T1 getFirst() const {
        return first;
    }

    T2 getSecond() const {
        return second;
    }
};

int main() {

    Pair<int, double> intDoublePair;
    intDoublePair.setValues(1, 2.5);
    cout << "First: " << intDoublePair.getFirst() << ", Second: " <<
    intDoublePair.getSecond() << endl;

    Pair<string, char> stringCharPair;
    stringCharPair.setValues("Hello", 'A');
    cout << "First: " << stringCharPair.getFirst() << ", Second: " <<
    stringCharPair.getSecond() << endl;

    return 0;
}
```

10. Pointer to Objects:

Define a class Book with data members title and author. Create an array of pointers to Book objects. Write functions to input details for each book, display the details, and search for a book by title.

```
class Book {
private:
    string title;
    string author;
public:
    void setDetails(const string& t, const string& a) {
        title = t;
        author = a;
    }

    void display() const {
        cout << "Title: " << title << ", Author: " << author << endl;
    }

    const string& getTitle() const {
        return title;
    }
};

void inputDetails(Book* books, int count) {
    for (int i = 0; i < count; ++i) {
        string title, author;
        cout << "Enter title and author for book " << i + 1 << ": ";
        cin.ignore(); // To clear the newline character from the input buffer
        getline(cin, title);
        getline(cin, author);
        books[i].setDetails(title, author);
    }
}

void displayDetails(Book* books, int count) {
    for (int i = 0; i < count; ++i) {
        books[i].display();
    }
}

Book* searchBookByTitle(Book* books, int count, const string& title) {
    for (int i = 0; i < count; ++i) {
        if (books[i].getTitle() == title) {
            return &books[i];
        }
    }
    return nullptr;
}

int main() {
    int numBooks;
    cout << "Enter the number of books: ";
    cin >> numBooks;
    Book* books = new Book[numBooks];
    inputDetails(books, numBooks);

    cout << "Book details:" << endl;
    displayDetails(books, numBooks);

    string searchTitle;
    cout << "Enter the title of the book to search: ";
    cin.ignore(); // To clear the newline character from the input buffer
    getline(cin, searchTitle);

    Book* foundBook = searchBookByTitle(books, numBooks, searchTitle);
    if (foundBook != nullptr) {
        cout << "Book found:" << endl;
        foundBook->display();
    } else {
        cout << "Book not found" << endl;
    }

    delete[] books;
    return 0;
}
```