FINAL TERM LAB TASKS

REPORT

NAME **M.Ismail**

REG NO **SU-24-01-001-054**

SECTION **CS & CYB**

SEMISTER **2nd**

DATE **25/08/2025**

SUBMITTED TO **MR NIAMAT ULLAH SEB**

Task#1:

Single Inheritance - Library System Problem Description: Create a base class Book with attributes title and author, and a method display(). Derive a class LibraryBook from Book that adds an attribute libraryID and overrides the display() method to include the library ID. Class Descriptions: • Book: This base class represents a book with title and author as protected attributes. It has a public method setDetails() to set these attributes and a public method display() to print them. • LibraryBook: This derived class represents a library book and inherits from Book. It adds a private attribute libraryID. It provides a public method setLibraryID() to set the libraryID and overrides the display() method to include the library ID. Main Function Details: • Create an object of the LibraryBook class. • Set the details for the LibraryBook object using the setDetails() and setLibraryID() methods. • Call the display() method to print the book details along with the library ID.

PROGRAM:

Task#01.M.Ismail-054.cpp

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

class Book {
protected:
    string title;
    string author;
public:
    void setDetails(string t, string a) {
        title = t;
        author = a;
    }
    virtual void display() {
        cout << "Title: " << title << endl;
        cout << "Author: " << author << endl;
    }
};

class LibraryBook : public Book {
private:
    int libraryID;
public:
    void setLibraryID(int id) {
        libraryID = id;
    }
    void display() override {
        cout << "Library Book Details:" << endl;
        cout << "Title: " << title << endl;
        cout << "Author: " << author << endl;
        cout << "Library ID: " << libraryID << endl;
    }
};

int main() {
    LibraryBook lb;
    lb.setDetails("The Great Gatsby", "F. Scott Fitzgerald");
    lb.setLibraryID(101);
    lb.display();
    return 0;
}
```

OUTPUT:

```
Library Book Details:
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Library ID: 101

--------------------------------
```

Task #2: Multiple Inheritance - Smart Device Problem Description: Create two base classes Camera and GPS. The Camera class should have a method takePhoto() that prints "Photo taken", and the GPS class should have a method showLocation() that prints "Location shown". Then, create a derived class Smartphone that inherits from both Camera and GPS. The Smartphone class should be able to take photos and show locations. Class Descriptions: • Camera: This base class represents a camera. It has a public method takePhoto() that prints "Photo taken". • GPS: This base class represents a GPS device. It has a public method showLocation() that prints "Location shown". • Smartphone: This derived class represents a smartphone that inherits from both Camera and GPS. It can use the functionality of both base classes. Main Function Details: • Create an object of the Smartphone class. • Call the takePhoto() method to simulate taking a photo. • Call the showLocation() method to simulate showing the location.

PROGRAM:

Task#02.M.Ismail-054.cpp

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

class Camera {
public:
    void takePhoto() {
        cout << "Photo taken" << endl;
    }
};

class GPS {
public:
    void showLocation() {
        cout << "Location shown" << endl;
    }
};

class Smartphone : public Camera, public GPS {
};

int main() {
    Smartphone s;
    s.takePhoto();
    s.showLocation();
    return 0;
}
```

OUTPUT:

```
Photo taken
Location shown

----------------
```

Task #3:

Hierarchical Inheritance - School System Problem Description: Create a base class Person with attributes name and age. Derive classes Teacher and Student from it. Each derived class should have a method to print their specific roles. Additionally, the Teacher class should have an attribute subject and the Student class should have an attribute grade. Class Descriptions: • Person: This base class represents a person with protected attributes name and age. It has a public method setDetails() to set these attributes and a public method display() to print them. • Teacher: This derived class represents a teacher and inherits from Person. It adds a private attribute subject. It provides a public method setSubject() to set the subject and a public method role() to print the teacher's role. • Student: This derived class represents a student and inherits from Person. It adds a private attribute grade. It provides a public method setGrade() to set the grade and a public method role() to print the student's role. Main Function Details: • Create an object of the Teacher class. • Set the details and subject for the Teacher object using the setDetails() and setSubject() methods. • Call the display() and role() methods to print the teacher's details and role. • Create an object of the Student class. • Set the details and grade for the Student object using the setDetails() and setGrade() methods. • Call the display() and role() methods to print the student's details and role.

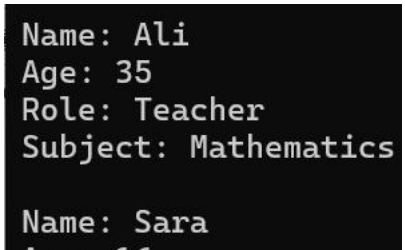
PROGRAM:

Task#03.M.Ismail-054.cpp

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

class Person {
protected:
    string name;
    int age;
public:
    void setDetails(string n, int a) {
        name = n;
        age = a;
    }
    void display() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

class Teacher : public Person {
private:
    string subject;
public:
    void setSubject(string s) {
        subject = s;
    }
    void role() {
        cout << "Role: Teacher" << endl;
        cout << "Subject: " << subject << endl;
    }
};

class Student : public Person {
private:
    string grade;
public:
    void setGrade(string g) {
        grade = g;
    }
    void role() {
        cout << "Role: Student" << endl;
        cout << "Grade: " << grade << endl;
    }
};
```

```cpp
int main() {
    Teacher t;
    t.setDetails("Ali", 35);
    t.setSubject("Mathematics");
    t.display();
    t.role();

    cout << endl;

    Student s;
    s.setDetails("Sara", 16);
    s.setGrade("10th");
    s.display();
    s.role();

    return 0;
}
```

OUTPUT:



Task #4: **Multilevel Inheritance - Electronic Devices Problem Description: Create a base class Device with an attribute brand. Derive a class Laptop from Device, and then derive a class GamingLaptop from Laptop. Each class should have a method to print details specific to their level. Class Descriptions: • Device: This base class represents a device with a protected attribute brand. It has a public method setBrand() to set the brand and a public method showBrand() to print the brand. • Laptop: This derived class represents a laptop and inherits from Device. It has a public method showType() to print the type of device. • GamingLaptop: This derived class represents a gaming laptop and inherits from Laptop. It has a public method showFeatures() to print the features of the gaming laptop. Main Function Details: • Create an object of the GamingLaptop class. • Set the brand for the GamingLaptop object using the setBrand() method. • Call the showBrand(), showType(), and showFeatures() methods to print the gaming laptop's brand, type, and features.**

PROGRAM:

Task#04.M.Ismail-054.cpp

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

class Device {
protected:
    string brand;
public:
    void setBrand(string b) {
        brand = b;
    }
    void showBrand() {
        cout << "Brand: " << brand << endl;
    }
};

class Laptop : public Device {
public:
    void showType() {
        cout << "Type: Laptop" << endl;
    }
};

class GamingLaptop : public Laptop {
public:
    void showFeatures() {
        cout << "Features: High-performance GPU, RGB Keyboard, Advanced Cooling System" << endl;
    }
};

int main() {
    GamingLaptop g;
    g.setBrand("ASUS ROG");
    g.showBrand();
    g.showType();
    g.showFeatures();
    return 0;
}
```

OUTPUT:

```
Brand: ASUS ROG
Type: Laptop
Features: High-performance GPU, RGB Keyboard, Advanced Cooling System
```

Lab Task #5: **Hybrid Inheritance - Vehicle System problem Description: Create a base class Vehicle with attributes make and year. Derive classes Car and Truck from Vehicle. Then, derive a class ElectricTruck from both Car and Truck using virtual inheritance to avoid ambiguity. Implement methods to display details specific to each class. Class Descriptions: • Vehicle: This base class represents a vehicle with protected attributes make and year. It has a public method setDetails() to set these attributes and a public method showDetails() to print them. • Car: This derived class represents a car and inherits from Vehicle using virtual inheritance. It has a public method carType() to print the type of vehicle. • Truck: This derived class represents a truck and inherits from Vehicle using virtual inheritance. It has a public method truckType() to print the type of vehicle. • ElectricTruck: This derived class represents an electric truck and inherits from both Car and Truck. It has a public method showElectricTruckDetails() to print the details of the electric truck, including the base and derived class attributes and methods. Main Function Details: • Create an object of the ElectricTruck class. • Set the details for the ElectricTruck object using the setDetails() method. • Call the showDetails() method to print the vehicle's make and year. • Call the carType() method to print the type of the vehicle as a car. • Call the truckType() method to print the type of the vehicle as a truck. • Call the showElectricTruckDetails() method to print the specific details of the electric truck.**

PROGRAM:

Task#05.M.Ismail-054.cpp

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

class Vehicle {
protected:
    string make;
    int year;
public:
    void setDetails(string m, int y) {
        make = m;
        year = y;
    }
    void showDetails() {
        cout << "Make: " << make << endl;
        cout << "Year: " << year << endl;
    }
};

class Car : virtual public Vehicle {
public:
    void carType() {
        cout << "Type: Car" << endl;
    }
};

class Truck : virtual public Vehicle {
public:
    void truckType() {
        cout << "Type: Truck" << endl;
    }
};

class ElectricTruck : public Car, public Truck {
public:
    void showElectricTruckDetails() {
        cout << "Electric Truck Features: Battery Powered, Zero Emissions, High Torque" << endl;
    }
};

int main() {
    ElectricTruck et;
    et.setDetails("Tesla", 2025);
    et.showDetails();
    et.carType();
    et.truckType();
    et.showElectricTruckDetails();
    return 0;
}
```

OUTPUT

```
Make: Tesla
Year: 2025
Type: Car
Type: Truck
Electric Truck Features: Battery Powered, Zero Emissions, High Torque
```

Task #6: :Vehicle Management System Using Inheritance and Constructors Problem Description: You are tasked with developing a program to manage different types of vehicles, specifically cars and bicycles. Each vehicle has a brand name and a year of manufacture. Cars have additional attributes such as the number of doors, whereas bicycles have attributes like the number of gears. Implement constructors to initialize these attributes for both vehicle types and demonstrate their usage in a main program. Program Implementation: Vehicle Class: Details: • Attributes: o brand (string): Represents the brand name of the vehicle. o year (int): Represents the year of manufacture of the vehicle. • Constructor: o Vehicle(string b, int y): Initializes the brand and year attributes with the provided values b and y. • Destructor: o ~Vehicle(): Prints a message indicating when a Vehicle object is being destroyed. • Member Function: o void display() const: Prints the brand and year of the vehicle. Car Class: Details: • Inherits from: Vehicle • Additional Attributes: o numDoors (int): Represents the number of doors the car has. • Constructor: o Car(string b, int y, int doors): Initializes the brand, year (via base class constructor), and numDoors with the provided values b, y, and doors. • Destructor: o ~Car(): Prints a message indicating when a Car object is being destroyed. • Member Function: o void display(): Overrides the display() function from Vehicle to include numDoors in the output. Bicycle Class: Details: • Inherits from: Vehicle • Additional Attributes: o numGears (int): Represents the number of gears the bicycle has. • Constructor: o Bicycle(string b, int y, int gears): Initializes the brand, year (via base class constructor), and numGears with the provided values b, y, and gears. • Destructor: o ~Bicycle(): Prints a message indicating when a Bicycle object is being destroyed. • Member Function: o void display(): Overrides the display() function from Vehicle to include numGears in the output. Main Function: Purpose: • Creates instances of Car and Bicycle. • Demonstrates usage of constructors, inheritance, . Steps: 1. Creates a Car object (myCar) with brand "Toyota", manufactured in 2022, and having 4 doors. 2. Creates a Bicycle object (myBike) with brand "Trek", manufactured in 2021, and having 18 gears. 3. Calls the display() function on both myCar and myBike to print their details, utilizing polymorphism to show different outputs based on the actual object type. Output: • Displays information about each vehicle, including brand, year, and specific attributes (numDoors for Car and numGears for Bicycle).

PROGRAM:

Task#06.M.Ismail-054.cpp

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

class Vehicle {
protected:
    string brand;
    int year;
public:
    Vehicle(string b, int y) : brand(b), year(y) {
        cout << "Vehicle created: " << brand << " (" << year << ")" << endl;
    }
    virtual ~Vehicle() {
        cout << "Vehicle destroyed: " << brand << endl;
    }
    virtual void display() const {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};

class Car : public Vehicle {
private:
    int numDoors;
public:
    Car(string b, int y, int doors) : Vehicle(b, y), numDoors(doors) {
        cout << "Car created: " << brand << endl;
    }
    ~Car() {
        cout << "Car destroyed: " << brand << endl;
    }
    void display() const override {
        cout << "Car - Brand: " << brand << ", Year: " << year << ", Doors: " << numDoors << endl;
    }
};

class Bicycle : public Vehicle {
private:
    int numGears;
public:
    Bicycle(string b, int y, int gears) : Vehicle(b, y), numGears(gears) {
        cout << "Bicycle created: " << brand << endl;
    }
    ~Bicycle() {
        cout << "Bicycle destroyed: " << brand << endl;
    }
    void display() const override {
        cout << "Bicycle - Brand: " << brand << ", Year: " << year << ", Gears: " << numGears << endl;
    }
};
```

```
int main() {
    Vehicle* myCar = new Car("Toyota", 2022, 4);
    Vehicle* myBike = new Bicycle("Trek", 2021, 18);

    myCar->display();
    myBike->display();

    delete myCar;
    delete myBike;

    return 0;
}
```

OUTPUT

```
Vehicle created: Toyota (2022)
Car created: Toyota
Vehicle created: Trek (2021)
Bicycle created: Trek
Car - Brand: Toyota, Year: 2022, Doors: 4
Bicycle - Brand: Trek, Year: 2021, Gears: 18
Car destroyed: Toyota
Vehicle destroyed: Toyota
Bicycle destroyed: Trek
Vehicle destroyed: Trek
```

Task #7: Pointer-Based Simple Calculator Objective: To apply pointer concepts in C++ by creating a simple calculator that performs arithmetic operations on two variables using pointers. Task Description: Create a program that declares two variables of different types (int, float, double). Use pointers to store the addresses of these variables. Implement a simple calculator that performs addition, subtraction, multiplication, and division on these variables using pointers. Display the results of the arithmetic operations. Implementation Steps: 1. Declare variables of types int, float, and double. 2. Declare pointers for each of these variables and initialize them with the addresses of the variables. 3. Implement functions for addition, subtraction, multiplication, and division that take pointers to the variables as parameters. 4. Call these functions and print the results.

PROGRAM:

Task#07.M.Ismail-054.cpp

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

int add(int* a, int* b) { return *a + *b; }
int subtract(int* a, int* b) { return *a - *b; }
int multiply(int* a, int* b) { return (*a) * (*b); }
float divide(int* a, int* b) { return (float)(*a) / (*b); }

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) { return (*a) / (*b); }

double add(double* a, double* b) { return *a + *b; }
double subtract(double* a, double* b) { return *a - *b; }
double multiply(double* a, double* b) { return (*a) * (*b); }
double divide(double* a, double* b) { return (*a) / (*b); }

int main() {
    int x = 10, y = 5;
    float p = 12.5, q = 4.5;
    double m = 20.75, n = 3.25;

    int* ip1 = &x; int* ip2 = &y;
    float* fp1 = &p; float* fp2 = &q;
    double* dp1 = &m; double* dp2 = &n;

    cout << "Integer Operations:" << endl;
    cout << "Addition: " << add(ip1, ip2) << endl;
    cout << "Subtraction: " << subtract(ip1, ip2) << endl;
    cout << "Multiplication: " << multiply(ip1, ip2) << endl;
    cout << "Division: " << divide(ip1, ip2) << endl << endl;

    cout << "Float Operations:" << endl;
    cout << "Addition: " << add(fp1, fp2) << endl;
    cout << "Subtraction: " << subtract(fp1, fp2) << endl;
    cout << "Multiplication: " << multiply(fp1, fp2) << endl;
    cout << "Division: " << divide(fp1, fp2) << endl << endl;

    cout << "Double Operations:" << endl;
    cout << "Addition: " << add(dp1, dp2) << endl;
    cout << "Subtraction: " << subtract(dp1, dp2) << endl;
    cout << "Multiplication: " << multiply(dp1, dp2) << endl;
    cout << "Division: " << divide(dp1, dp2) << endl;

    return 0;
}
```

OUTPUT:

```
Integer Operations:
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2

Float Operations:
Addition: 17
Subtraction: 8
Multiplication: 56.25
Division: 2.77778

Double Operations:
Addition: 24
Subtraction: 17.5
Multiplication: 67.4375
Division: 6.38462
```

TASK#8: Pointer-Based Swapping of Values Objective: To practice using pointers for swapping the values of two variables in C++. Task Description: Create a program that declares two integer variables. Use pointers to swap the values of these two variables. Display the values before and after the swap. Implementation Steps: 1. Declare two integer variables and initialize them with different values. 2. Declare pointers to these integer variables. 3. Implement a function to swap

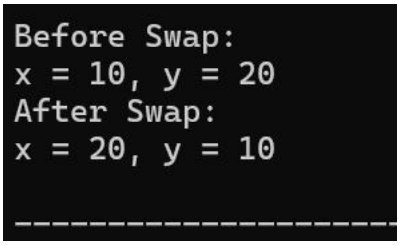
the values of the variables using the pointers. 4. Call the swap function and print the values before and after the swap.

PROGRAM:

Task#08.M.Ismail-054.cpp

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

void swapValues(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 10, y = 20;
    int* p1 = &x;
    int* p2 = &y;

    cout << "Before Swap:" << endl;
    cout << "x = " << x << ", y = " << y << endl;

    swapValues(p1, p2);

    cout << "After Swap:" << endl;
    cout << "x = " << x << ", y = " << y << endl;

    return 0;
}
```

OUTPUT:



TASK#9: Basic Encapsulation Implementation Objective: Create a basic class to practice implementing encapsulation. Instructions: 1. Create a Class: o Define a class named Book with private data members for title, author, and price. 2. Implement Setter Methods: o Implement public setter methods to set the values of title, author, and price. 3. Implement Getter Methods: o Implement public

getter methods to retrieve the values of title, author, and price. 4. Demonstrate Usage: o In the main function, create an object of Book. o Use the setter methods to set the values and the getter methods to display them.

PROGRAM:

Task#09.M.Ismail-054.cpp

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

class Book {
private:
    string title;
    string author;
    float price;
public:
    void setTitle(string t) { title = t; }
    void setAuthor(string a) { author = a; }
    void setPrice(float p) { price = p; }

    string getTitle() { return title; }
    string getAuthor() { return author; }
    float getPrice() { return price; }
};

int main() {
    Book b;
    b.setTitle("The Great Gatsby");
    b.setAuthor("F. Scott Fitzgerald");
    b.setPrice(499.99);

    cout << "Book Details:" << endl;
    cout << "Title: " << b.getTitle() << endl;
    cout << "Author: " << b.getAuthor() << endl;
    cout << "Price: " << b.getPrice() << endl;

    return 0;
}
```

OUTPUT:

```
Book Details:
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Price: 499.99

--------------------------------
```

TASK#10: Encapsulation with Validation Objective: Implement a class with encapsulation and validation. Instructions: 1. Create a Class: o Define a class named Student with private data members for name, age, and grade. 2. Implement Setter Methods with Validation: o Implement public setter methods to set the values of name, age, and grade with validation (e.g., age must be non-negative, grade must be between 0 and 100). 3. Implement Getter Methods: o Implement public getter methods to retrieve the values of name, age, and grade. 4. Demonstrate Usage: o In the main function, create an object of Student. o Use the setter methods to set the values and the getter methods to display them, including handling invalid inputs.

PROGRAM :

Task#10.M.Ismail-054.cpp

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

class Student {
private:
    string name;
    int age;
    int grade;
public:
    void setName(string n) {
        if (n.empty()) {
            cout << "Invalid name. Setting default name 'Unknown'." << endl;
            name = "Unknown";
        } else {
            name = n;
        }
    }
    void setAge(int a) {
        if (a < 0) {
            cout << "Invalid age. Setting age to 0." << endl;
            age = 0;
        } else {
            age = a;
        }
    }
    void setGrade(int g) {
        if (g < 0 || g > 100) {
            cout << "Invalid grade. Setting grade to 0." << endl;
            grade = 0;
        } else {
            grade = g;
        }
    }

    string getName() { return name; }
    int getAge() { return age; }
    int getGrade() { return grade; }
};

int main() {
    Student s;

    s.setName("Ali");
    s.setAge(20);
    s.setGrade(95);

    cout << "Student Details:" << endl;
    cout << "Name: " << s.getName() << endl;
    cout << "Age: " << s.getAge() << endl;
    cout << "Grade: " << s.getGrade() << endl;

    cout << endl << "Testing invalid inputs..." << endl;
    s.setName("");
    s.setAge(-5);
    s.setGrade(150);

    cout << "Student Details After Invalid Inputs:" << endl;
    cout << "Name: " << s.getName() << endl;
    cout << "Age: " << s.getAge() << endl;
    cout << "Grade: " << s.getGrade() << endl;

    return 0;
}
```

OUTPUT:

```
Student Details:
Name: Ali
Age: 20
Grade: 95

Testing invalid inputs...
Invalid name. Setting default name 'Unknown'
Invalid age. Setting age to 0.
Invalid grade. Setting grade to 0.
Student Details After Invalid Inputs:
Name: Unknown
Age: 0
Grade: 0
```

Lab Task 11: Animal Sounds Objective: Implement run-time polymorphism to manage different types of animals and their sounds. Instructions: 1. Base Class: Create a class Animal with a pure virtual function makeSound(). 2. Derived Classes: Create two derived classes, Dog and Cat, that override the makeSound() function to print appropriate sounds ("Woof! Woof!" for dogs and "Meow! Meow!" for cats). 3. Main Function: o Create objects of Dog and Cat. o Use a base class pointer to call the makeSound() function for both types of animals. Expected Output: Woof! Woof! Meow! Meow!

PROGRAM:

Task#11.M.Ismail-054.cpp

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

class Animal {
public:
    virtual void makeSound() = 0;
};

class Dog : public Animal {
public:
    void makeSound() override {
        cout << "Woof! Woof!" << endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {
        cout << "Meow! Meow!" << endl;
    }
};

int main() {
    Animal* a1 = new Dog();
    Animal* a2 = new Cat();

    a1->makeSound();
    a2->makeSound();

    delete a1;
    delete a2;

    return 0;
}
```

OUTPUT:

```
Woof! Woof!
Meow! Meow!
```

Lab Task #12 : Vehicle Descriptions Objective: Implement run-time polymorphism to describe different types of vehicles. Instructions: 1. Base Class: Create a class Vehicle with a pure virtual function describe(). 2. Derived Classes: o Create a class Car that overrides describe() to print the car's make and model. o Create a class Bicycle that overrides describe() to print the number of gears. 3. Main Function: o Create objects of Car and Bicycle. o Use a base class pointer to call the describe() function for both types of vehicles. Expected Output: This is a car. Make: Toyota, Model: Corolla This is a bicycle with 18 gears.

PROGRAM:

Task#12.M.Ismail-054.cpp

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

class Vehicle {
public:
    virtual void describe() = 0;
};

class Car : public Vehicle {
private:
    string make;
    string model;
public:
    Car(string m, string mod) : make(m), model(mod) {}
    void describe() override {
        cout << "This is a car. Make: " << make << ", Model: " << model << endl;
    }
};

class Bicycle : public Vehicle {
private:
    int gears;
public:
    Bicycle(int g) : gears(g) {}
    void describe() override {
        cout << "This is a bicycle with " << gears << " gears." << endl;
    }
};

int main() {
    Vehicle* v1 = new Car("Toyota", "Corolla");
    Vehicle* v2 = new Bicycle(18);

    v1->describe();
    v2->describe();

    delete v1;
    delete v2;

    return 0;
}
```

OUTPUT:

```
This is a car. Make: Toyota, Model: Corolla
This is a bicycle with 18 gears.
```

Lab Task 13:

: Notification System Objective: Implement run-time polymorphism to handle and send different types of notifications.

Instructions: 1. Base Class: Create a class Notification with a pure virtual function send(). 2. Derived Classes: o Create a class EmailNotification that overrides send() to print a message about sending an email. o Create a class SMSNotification that overrides send() to print a message about sending an SMS. 3. Main Function: o Create objects of EmailNotification and SMSNotification. o Use a base class pointer to call the send() function for both types of notifications. Expected Output: Sending email to: example@example.com Sending SMS to: 123-456-7890

Program:

Task#13.M.Ismail-054.cpp

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

class Notification {
public:
    virtual void send() = 0;
};

class EmailNotification : public Notification {
private:
    string email;
public:
    EmailNotification(string e) : email(e) {}
    void send() override {
        cout << "Sending email to: " << email << endl;
    }
};

class SMSNotification : public Notification {
private:
    string phoneNumber;
public:
    SMSNotification(string p) : phoneNumber(p) {}
    void send() override {
        cout << "Sending SMS to: " << phoneNumber << endl;
    }
};

int main() {
    Notification* n1 = new EmailNotification("example@example.com");
    Notification* n2 = new SMSNotification("123-456-7890");

    n1->send();
    n2->send();

    delete n1;
    delete n2;

    return 0;
}
```

OUTPUT:

```
Sending email to: example@example.com
Sending SMS to: 123-456-7890
```

Lab Task 14:

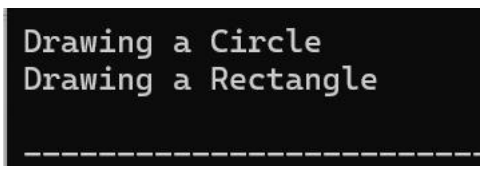
Lecture Outlines 1. Introduction to Abstraction 2. How Abstraction Works in C++ 3. Examples and Analogies 4. Simplified Example with Abstraction Using an Abstract Class 5. Demonstrating Abstraction with a C++ Interface Class Lab Objectives: • Understand the concept of abstraction in OOP using C++. • Implement abstraction using abstract classes in C++. • Implement abstraction using interfaces in C++. • Demonstrate the use of abstraction through practical examples and coding exercises

Program:

Task#14.M.Ismail-054.cpp

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

class Shape {
public:
    virtual void draw() = 0; // Pure virtual function
};

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

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

int main() {
    Shape* s1 = new Circle();
    Shape* s2 = new Rectangle();

    s1->draw();
    s2->draw();

    delete s1;
    delete s2;
    return 0;
}
```

OUTPUT:



Lab Task 15: Basic Abstract Class Implementation Objective: Create a basic abstract class and derived classes to practice implementing abstraction. Instructions: 1. Create an Abstract Class: o Define an abstract class named Appliance with a pure virtual function turnOn(). 2. Implement Derived Classes: o Derive two classes from Appliance: Fan and Light. o Implement the turnOn() function in both Fan and Light classes to print specific messages (e.g., "Fan is now on" and "Light is now on"). 3. Demonstrate Usage: o In the main function, create objects of Fan and Light. o Use pointers of type Appliance to call the turnOn() function for both objects.

Program:

Task#15.M.Ismail-054.cpp

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

class Appliance {
public:
    virtual void turnOn() = 0;
};

class Fan : public Appliance {
public:
    void turnOn() override {
        cout << "Fan is now on" << endl;
    }
};

class Light : public Appliance {
public:
    void turnOn() override {
        cout << "Light is now on" << endl;
    }
};

int main() {
    Appliance* a1 = new Fan();
    Appliance* a2 = new Light();

    a1->turnOn();
    a2->turnOn();

    delete a1;
    delete a2;

    return 0;
}
```

OUTPUT:

```
Fan is now on
Light is now on
```

Lab Task 16:

Simple Voting Eligibility Check Objective: Practice handling exceptions based on user input for age validation. Description: 1. Write a C++ program that checks if a person is eligible to vote based on their age. 2. If the age entered by the user is less than 18, throw a custom exception indicating ineligibility to vote. 3. Handle this exception to display a user-friendly error message. Instructions: 1. Setup: o Include necessary headers: for input/output and for standard exception handling. o Define a custom exception class NotEligibleToVoteException inheriting from exception. Override the what() method to return a specific error message. 2. Implementation: o Prompt the user to enter their age. o Use a try block to check if the age is below 18. o If the condition is met, throw an instance of NotEligibleToVoteException. o In the catch block, handle the exception and display the error message. 3. Testing: o Run the program and enter various ages (e.g., 16, 18, 25) to verify it correctly identifies whether the user is eligible to vote or not.

Program:

Task#16.M.Ismail-054.cpp

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

class NotEligibleToVoteException : public exception {
public:
    const char* what() const noexcept override {
        return "Error: You are not eligible to vote (must be 18 or older).";
    }
};

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;

    try {
        if (age < 18) {
            throw NotEligibleToVoteException();
        }
        cout << "You are eligible to vote!" << endl;
    }
    catch (const NotEligibleToVoteException& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

OUTPUT:

```
Enter your age: 18
You are eligible to vote!
```

Lab Task 17: Simple Password Validation Objective: Handle exceptions for validating password length.

Description: 1. Write a C++ program that checks if a password meets the minimum length requirement (e.g., 6 characters). 2. If the entered password is shorter than the required length, throw a custom exception for invalid password length. 3. Handle this exception to display an appropriate message to the user. Instructions: 1. Setup: o Include necessary headers: for input/output and for standard exception handling. o Define a custom exception class ShortPasswordException inheriting from exception. Override the what() method to return a specific error message. 2. Implementation: o Prompt the user to enter a password. o Use a try block to check if the password length is less than 6 characters. o If the condition is met, throw an instance of ShortPasswordException. o In the catch block, handle the exception and display the error message. 3. Testing: o Run the program and test with various passwords (e.g., "abc",

"password123") to ensure it correctly validates the password length.

Program:

Task#17.M.Ismail-054.cpp

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

class ShortPasswordException : public exception {
public:
    const char* what() const noexcept override {
        return "Error: Password too short (minimum length is 6 characters).";
    }
};

int main() {
    string password;
    cout << "Enter your password: ";
    cin >> password;

    try {
        if (password.length() < 6) {
            throw ShortPasswordException();
        }
        cout << "Password accepted!" << endl;
    }
    catch (const ShortPasswordException& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

OUTPUT:

```
Enter your password: 133741
Password accepted!
```

Lab Task 18:

Simple Number Squaring Objective: Handle exceptions when parsing non-numeric input and calculate the square of a number. Description: 1. Write a C++ program that reads a string input from the user and attempts to parse it as a number. 2. If the input is not a valid number (contains non-numeric characters), throw an exception and handle it to display an error message. 3. If the input is valid, compute and display the square of the number. Instructions: 1. Setup: o Include necessary headers: for input/output, for string operations, and for exception handling. o Define a custom exception class InvalidInputException inheriting from exception. Override the what() method to return a specific error message. 2. Implementation: o Prompt the user to enter a number. o Use a try block to convert the input string to a numeric value. o If the conversion fails or if extra characters are present, throw an InvalidInputException. o In the catch block, handle the exception and display the error message. 3. Testing: o Run the program and test with various inputs, including valid numbers (e.g., "5", "3.14") and invalid inputs (e.g., "abc", "12abc") to verify it correctly handles the input and computes the square.

Program:

Task#18.M.Ismail-054.cpp

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

class InvalidInputException : public exception {
public:
    const char* what() const noexcept override {
        return "Error: Invalid input. Please enter a valid number.";
    }
};

int main() {
    string input;
    cout << "Enter a number: ";
    cin >> input;

    try {
        size_t pos;
        double number = stod(input, &pos);

        if (pos != input.length()) {
            throw InvalidInputException();
        }

        cout << "Square: " << (number * number) << endl;
    }
    catch (const InvalidInputException& e) {
        cout << e.what() << endl;
    }
    catch (const invalid_argument&) {
        cout << "Error: Invalid input. Please enter a valid number." << endl;
    }
    catch (const out_of_range&) {
        cout << "Error: Number out of range." << endl;
    }

    return 0;
}
```

OUTPUT:

```
Enter a number: 3
Square: 9

--------------------------------
```

Lab Task 19: **Writing Student Information to a File Objective To understand how to use classes and objects in C++ to write data into a file. Task Description 1. Create a class Student that has the following private data members: o string name o int age o string grade 2. Add public member functions to the class: o A constructor to initialize the data members. o A member function void getInput() to input data from the user. o A member function void writeToFile() to write the student data to a file. 3. In the main function: o Create an object of the Student class. o Use the getInput() function to get the student information from the user. o Use the writeToFile() function to write the information to a file named student_data.txt.**

Program:

Task#19.M.Ismail-054.cpp

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

class Student {
private:
    string name;
    int age;
    string grade;

public:
    Student() : age(0) {}

    void getInput() {
        cout << "Enter name: ";
        getline(cin, name);
        cout << "Enter age: ";
        cin >> age;
        cin.ignore();
        cout << "Enter grade: ";
        getline(cin, grade);
    }

    void writeToFile() {
        ofstream file("student_data.txt", ios::app);
        if (file.is_open()) {
            file << "Name: " << name << endl;
            file << "Age: " << age << endl;
            file << "Grade: " << grade << endl;
            file << "------------------------" << endl;
            file.close();
            cout << "Data written to student_data.txt successfully!" << endl;
        } else {
            cout << "Error opening file!" << endl;
        }
    }
};

int main() {
    Student s;
    s.getInput();
    s.writeToFile();
    return 0;
}
```

OUTPUT:

```
Enter name: irfan
Enter age: 18
Enter grade: A
Data written to student_data.txt successfully!
```

Lab Task #20: Writing and Reading Student Information to/from a File Objective To understand how to use classes and objects in C++ to write data into a file and read data from a file. Task Description 1. Create a class Student that has the following private data members: o string name o int age o string grade 2. Add public member functions to the class: o A constructor to initialize the data members. o A member function void getInput() to input data from the user. o A member function void writeToFile() to write the student data to a file. o A member function void readFromFile() to read student data from the file and display it. 3. In the main function: o Create an object of the Student class. o Use the getInput() function to get the student information from the user. o Use the writeToFile() function to write the information to a file named student_data.txt. o Use the readFromFile() function to read and display the student information from the file.

Program:

Task#20.M.Ismail-054.cpp

```cpp
public:
    Student() : age(0) {}

    void getInput() {
        cout << "Enter name: ";
        getline(cin, name);
        cout << "Enter age: ";
        cin >> age;
        cin.ignore();
        cout << "Enter grade: ";
        getline(cin, grade);
    }

    void writeToFile() {
        ofstream file("student_data.txt", ios::app);
        if (file.is_open()) {
            file << name << endl;
            file << age << endl;
            file << grade << endl;
            file.close();
            cout << "Data written to student_data.txt successfully!" << endl;
        } else {
            cout << "Error opening file for writing!" << endl;
        }
    }

    void readFromFile() {
        ifstream file("student_data.txt");
        if (file.is_open()) {
            string line;
            cout << "\nReading data from student_data.txt:\n";
            while (getline(file, line)) {
                cout << line << endl;
            }
            file.close();
        } else {
            cout << "Error opening file for reading!" << endl;
        }
    }
};

int main() {
    Student s;
    s.getInput();
    s.writeToFile();
    s.readFromFile();
    return 0;
}
```

OUTPUT:

```
Enter name: C++
Enter age: 10
Enter grade: B
Data written to student_data.txt successfully!

Reading data from student_data.txt:
Name: irfan
Age: 18
Grade: A
-----------------------
C++
10
B
```

Lab Task #21: Counting the Number of Lines in a File Objective To understand how to use classes and objects in C++ to perform file operations, specifically to count the number of lines in a file. Task Description 1. Create a class FileHandler that has the following private data members: o string filename 2. Add public member functions to the class: o A constructor to initialize the filename. o A member function void writeSampleData() to write some sample data to the file. o A member function int countLines() to count the number of lines in the file and return the count. 3. In the main function: o Create an object of the FileHandler class. o Use the writeSampleData() function to write some sample data to the file. o Use the countLines() function to count and display the number of lines in the file.

Program:

Task#21.M.Ismail-054.cpp

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

class FileHandler {
private:
    string filename;

public:
    FileHandler(string fname) : filename(fname) {}

    void writeSampleData() {
        ofstream file(filename);
        if (file.is_open()) {
            file << "This is line 1" << endl;
            file << "This is line 2" << endl;
            file << "This is line 3" << endl;
            file.close();
            cout << "Sample data written to " << filename << endl;
        } else {
            cout << "Error opening file for writing!" << endl;
        }
    }

    int countLines() {
        ifstream file(filename);
        int count = 0;
        string line;
        if (file.is_open()) {
            while (getline(file, line)) {
                count++;
            }
            file.close();
        } else {
            cout << "Error opening file for reading!" << endl;
        }
        return count;
    }
};

int main() {
    FileHandler fh("sample.txt");
    fh.writeSampleData();
    int lines = fh.countLines();
    cout << "Number of lines in file: " << lines << endl;
    return 0;
}
```

OUTPUT:

```
Sample data written to sample.txt
Number of lines in file: 3
```