ASSIGNMENT 5:

Name: Om Prasanna Kulkarni

PRN: 22510034

Batch: S3

Branch: CSE

Topic: Namespace and Virtual Functions

1. You are provided with two libraries. Both libraries have a class called LibraryClass and both have a function called show(). Your task is to utilize both of these classes in a main program without any naming conflicts.

SOURCE CODE:  
#include <iostream>

namespace Library1 {

    class LibraryClass {

    public:

        void show() {

            std::cout << "Library1::LibraryClass::show()" << std::endl;

        }

    };

    void show() {

        std::cout << "Library1::show()" << std::endl;

    }

}

namespace Library2 {

    class LibraryClass {

    public:

        void show() {

            std::cout << "Library2::LibraryClass::show()" << std::endl;

        }

    };

    void show() {

        std::cout << "Library2::show()" << std::endl;

    }

}

int main() {

    Library1::LibraryClass obj1;

    Library2::LibraryClass obj2;

    Library1::show();

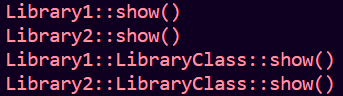
    Library2::show();

    obj1.show();

    obj2.show();

    return 0;

}

OUTPUT:  


1. You are developing software for a robotic company. The company manufactures ground robots and drones. Each type of robot has different components. Your task is to define nested namespaces that categorize these robots and their components.

SOURCE CODE:

#include <iostream>

namespace RoboCompany {

    namespace GroundRobots {

        namespace Types {

            class WheeledRobot {

            public:

                void drive() {

                    std::cout << "Wheeled robot is driving." << std::endl;

                }

            };

            class LeggedRobot {

            public:

                void walk() {

                    std::cout << "Legged robot is walking." << std::endl;

                }

            };

        }

        namespace Components {

            class Camera {

            public:

                void captureImage() {

                    std::cout << "Camera is capturing an image." << std::endl;

                }

            };

            class Arm {

            public:

                void pickObject() {

                    std::cout << "Arm is picking up an object." << std::endl;

                }

            };

        }

    }

    namespace Drones {

        namespace Types {

            class Quadcopter {

            public:

                void fly() {

                    std::cout << "Quadcopter is flying." << std::endl;

                }

            };

            class Hexacopter {

            public:

                void hover() {

                    std::cout << "Hexacopter is hovering." << std::endl;

                }

            };

        }

        namespace Components {

            class GPS {

            public:

                void navigate() {

                    std::cout << "GPS is helping with navigation." << std::endl;

                }

            };

            class Camera {

            public:

                void recordVideo() {

                    std::cout << "Camera is recording video." << std::endl;

                }

            };

        }

    }

}

int main() {

    RoboCompany::GroundRobots::Types::WheeledRobot wheeledRobot;

    wheeledRobot.drive();

    RoboCompany::GroundRobots::Components::Camera camera;

    camera.captureImage();

    RoboCompany::Drones::Types::Quadcopter quadcopter;

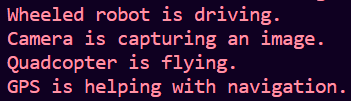
    quadcopter.fly();

    RoboCompany::Drones::Components::GPS gps;

    gps.navigate();

    return 0;

}

OUTPUT:  


1. You have to design a basic system where different animals make sounds. The base class Animal will have a function makeSound(), which should be overridden by derived classes to make specific sounds.

SOURCE CODE:  
#include <iostream>

class Animal {

public:

    virtual void makeSound() {

        std::cout << "The animal makes a generic sound." << std::endl;

    }

};

class Dog : public Animal {

public:

    void makeSound() override {

        std::cout << "The dog barks." << std::endl;

    }

};

class Cat : public Animal {

public:

    void makeSound() override {

        std::cout << "The cat meows." << std::endl;

    }

};

class Cow : public Animal {

public:

    void makeSound() override {

        std::cout << "The cow moos." << std::endl;

    }

};

int main() {

    Animal\* animal1 = new Dog();

    Animal\* animal2 = new Cat();

    Animal\* animal3 = new Cow();

    animal1->makeSound();

    animal2->makeSound();

    animal3->makeSound();

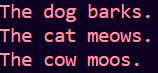
    delete animal1;

    delete animal2;

    delete animal3;

    return 0;

}

OUTPUT:  


1. You are to design a system where different shapes calculate their areas. The base class Shape will have a function area(), which should be overridden by derived classes to calculate specific areas.

SOURCE CODE:

#include <iostream>

class Shape {

public:

    virtual double area() const {

        std::cout << "Calculating the area of a generic shape." << std::endl;

        return 0.0;

    }

};

class Circle : public Shape {

private:

    double radius;

public:

    Circle(double r) : radius(r) {}

    double area() const override {

        return 3.14159265359 \* radius \* radius;

    }

};

class Rectangle : public Shape {

private:

    double width;

    double height;

public:

    Rectangle(double w, double h) : width(w), height(h) {}

    double area() const override {

        return width \* height;

    }

};

int main() {

    Shape\* shape1 = new Circle(5.0);

    Shape\* shape2 = new Rectangle(4.0, 6.0);

    std::cout << "Area of the circle: " << shape1->area() << std::endl;

    std::cout << "Area of the rectangle: " << shape2->area() << std::endl;

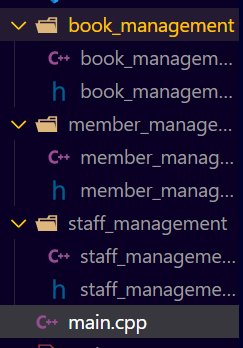
    delete shape1;

    delete shape2;

    return 0;

}

OUTPUT:  


Group Task: Building a Modular Library System with Namespaces Objective: Design a library management system using namespaces to organize various features. Background: A library management system has various modules - book management, member management, and staff management. Instructions: Divide task: Book Management: Manages adding, deleting, and searching for books. Member Management: Manages member registrations, deletions, and searching for members. Staff Management: Manages staff hiring, firing, and searching for staff. Develop Namespace Structured Header Files: book\_management.h by Book Management Team  
  
  
Source Code:  


Book Management.cpp

#include "book\_management.h"

#include <iostream>

#include <vector>

namespace BookManagement {

    std::vector<Book> bookDatabase;

    void addBook(Book& book) {

        bookDatabase.push\_back(book);

        std::cout << "Book added successfully." << std::endl;

    }

    void deleteBook(Book& book) {

        std::cout << "Book deleted successfully." << std::endl;

    }

    Book\* searchBook(const std::string& title) {

        for (auto& b : bookDatabase) {

            if (b.title == title) {

                return &b;

            }

        }

        return nullptr;

    }

}

Book Management.h

#ifndef BOOK\_MANAGEMENT\_H

#define BOOK\_MANAGEMENT\_H

#include <string>

#include <vector>

namespace BookManagement {

    struct Book {

        std::string title;

    };

    void addBook(Book& book);

    void deleteBook(Book& book);

    Book\* searchBook(const std::string& title);

}

#endif

Member Management.cpp

#include "member\_management.h"

#include <iostream>

#include <vector>

namespace MemberManagement {

    std::vector<Member> memberDatabase;

    void registerMember(Member& member) {

        memberDatabase.push\_back(member);

        std::cout << "Member registered successfully." << std::endl;

    }

    void deleteMember(Member& member) {

*/*

        std::cout << "Member deleted successfully." << std::endl;

    }

    Member\* searchMember(const std::string& name) {

        for (auto& m : memberDatabase) {

            if (m.name == name) {

                return &m;

            }

        }

        return nullptr;

    }

}

Member Management.h

#ifndef MEMBER\_MANAGEMENT\_H

#define MEMBER\_MANAGEMENT\_H

#include <string>

#include <vector>

namespace MemberManagement {

    struct Member {

        std::string name;

    };

    void registerMember(Member& member);

    void deleteMember(Member& member);

    Member\* searchMember(const std::string& name);

}

#endif

Staff Management.cpp

#include "staff\_management.h"

#include <iostream>

#include <vector>

namespace StaffManagement {

    std::vector<Staff> staffDatabase;

    void hireStaff(Staff& staff) {

        staffDatabase.push\_back(staff);

        std::cout << "Staff hired successfully." << std::endl;

    }

    void fireStaff(Staff& staff) {

        std::cout << "Staff member fired successfully." << std::endl;

    }

    Staff\* searchStaff(const std::string& name) {

        for (auto& s : staffDatabase) {

            if (s.name == name) {

                return &s;

            }

        }

        return nullptr;

    }

}

Staff Management.h

#ifndef STAFF\_MANAGEMENT\_H

#define STAFF\_MANAGEMENT\_H

#include <string>

#include <vector>

namespace StaffManagement {

    struct Staff {

        std::string name;

    };

    void hireStaff(Staff& staff);

    void fireStaff(Staff& staff);

    Staff\* searchStaff(const std::string& name);

}

#endif

Main.cpp

#include "book\_management/book\_management.h"

#include "member\_management/member\_management.h"

#include "staff\_management/staff\_management.h"

int main() {

    BookManagement::Book book;

    book.title = "Introduction to C++";

    BookManagement::addBook(book);

    MemberManagement::Member member;

    member.name = "John Doe";

    MemberManagement::registerMember(member);

    StaffManagement::Staff staff;

    staff.name = "Alice Smith";

    StaffManagement::hireStaff(staff);

    return 0;

}

OUTPUT:  
