**LAB PROGRAMS(26/06)**

Design a class hierarchy to represent various entities in a university system.

**1.**Base Class (Person):

Data members: name (string), age (int)

Member functions: getDetails(), a virtual function to print basic person details

Derived Class (Student): (Single Inheritance)

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

Person(const std::string& name, int age) : name(name), age(age) {}

virtual void getDetails() const {

std::cout << "Name: " << name << ", Age: " << age;

}

virtual ~Person() = default; };

class Student : public Person {

private:

std::string student\_id;

std::string major;

public:

Student(const std::string& name, int age, const std::string& student\_id, const std::string& major)

: Person(name, age), student\_id(student\_id), major(major) {}

void getDetails() const override {

Person::getDetails();

std::cout << ", Student ID: " << student\_id << ", Major: " << major;

}

};

int main() {

Person\* person = new Person("Alice", 45);

Person\* student = new Student("Bob", 20, "S123456", "Computer Science");

person->getDetails();

std::cout << std::endl;

student->getDetails();

std::cout << std::endl;

delete person;

delete student;

return 0;

}

OUTPUT:

Name: Alice, Age: 45

Name: Bob, Age: 20, Student ID: S123456, Major: Computer Science

**2**..Inherits from Person

Data members: studentId (int), major (string)

Member functions:

setMajor(string) to set the student's major

getMajor() to retrieve the major

Override getDetails() to include student-specific information

Derived Class (Faculty): (Single Inheritance)

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

Person(const std::string& name, int age) : name(name), age(age) {}

virtual void getDetails() const {

std::cout << "Name: " << name << ", Age: " << age;

}

virtual ~Person() = default;

};

class Student : public Person {

private:

int studentId;

std::string major;

public:

Student(const std::string& name, int age, int studentId, const std::string& major)

: Person(name, age), studentId(studentId), major(major) {}

void setMajor(const std::string& major) {

this->major = major;

}

std::string getMajor() const {

return major;

}

void getDetails() const override {

Person::getDetails();

std::cout << ", Student ID: " << studentId << ", Major: " << major;

}

};

class Faculty : public Person {

private:

std::string department;

std::string title;

public:

Faculty(const std::string& name, int age, const std::string& department, const std::string& title)

: Person(name, age), department(department), title(title) {}

void getDetails() const override {

Person::getDetails();

std::cout << ", Department: " << department << ", Title: " << title;

}

};

int main() {

Person\* person = new Person("Alice", 45);

Student\* student = new Student("Bob", 20, 123456, "Computer Science");

Faculty\* faculty = new Faculty("Dr. Smith", 50, "Engineering", "Professor");

person->getDetails();

std::cout << std::endl;

student->getDetails();

std::cout << std::endl;

faculty->getDetails();

std::cout << std::endl;

delete person;

delete student;

delete faculty;

return 0;

}

OUTPUT:

Name: Alice, Age: 45

Name: Bob, Age: 20, Student ID: 123456, Major: Computer Science

Name: Dr. Smith, Age: 50, Department: Engineering, Title: Professor

1. Inherits from Student (inherits indirectly from Person as well) Data member: coursesTeaching (array/vector of strings) Member functions: setCoursesTeaching(string[]) to set the courses the TA is teaching getCoursesTeaching() to retrieve the list of courses Override getDetails() to include TA-specific information (e.g., courses) Derived Class (ResearchAssistant): (Hierarchical Inheritance) in c++

#include <iostream>

#include <vector>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

Person(const std::string& name, int age) : name(name), age(age) {}

virtual void getDetails() const {

std::cout << "Name: " << name << ", Age: " << age;

}

virtual ~Person() = default;

};

class Student : public Person {

private:

int studentId;

std::string major;

public:

Student(const std::string& name, int age, int studentId, const std::string& major)

: Person(name, age), studentId(studentId), major(major) {}

void setMajor(const std::string& major) {

this->major = major;

}

std::string getMajor() const {

return major;

}

void getDetails() const override {

Person::getDetails();

std::cout << ", Student ID: " << studentId << ", Major: " << major;

}

};

class TeachingAssistant : public Student {

private:

std::vector<std::string> coursesTeaching;

public:

TeachingAssistant(const std::string& name, int age, int studentId, const std::string& major)

: Student(name, age, studentId, major) {}

void setCoursesTeaching(const std::vector<std::string>& courses) {

coursesTeaching = courses;

}

std::vector<std::string> getCoursesTeaching() const {

return coursesTeaching;

}

void getDetails() const override {

Student::getDetails();

std::cout << ", Courses Teaching: ";

for (const auto& course : coursesTeaching) {

std::cout << course << " ";

}

}

};

class ResearchAssistant : public Student {

private:

std::string researchArea;

std::string supervisor;

public:

ResearchAssistant(const std::string& name, int age, int studentId, const std::string& major,

const std::string& researchArea, const std::string& supervisor)

: Student(name, age, studentId, major), researchArea(researchArea), supervisor(supervisor) {}

void setResearchArea(const std::string& researchArea) {

this->researchArea = researchArea;

}

std::string getResearchArea() const {

return researchArea;

}

void setSupervisor(const std::string& supervisor) {

this->supervisor = supervisor;

}

std::string getSupervisor() const {

return supervisor;

}

void getDetails() const override {

Student::getDetails();

std::cout << ", Research Area: " << researchArea << ", Supervisor: " << supervisor;

}

};

int main() {

Person\* person = new Person("Alice", 45);

Student\* student = new Student("Bob", 20, 123456, "Computer Science");

TeachingAssistant\* ta = new TeachingAssistant("John", 25, 654321, "Mathematics");

ResearchAssistant\* ra = new ResearchAssistant("Charlie", 30, 789012, "Physics", "Quantum Computing", "Dr. Smith");

std::vector<std::string> courses = {"Calculus", "Algebra"};

ta->setCoursesTeaching(courses);

person->getDetails();

std::cout << std::endl;

student->getDetails();

std::cout << std::endl;

ta->getDetails();

std::cout << std::endl;

ra->getDetails();

std::cout << std::endl;

delete person;

delete student;

delete ta;

delete ra;

return 0;

}

OUTPUT:

Name: Alice, Age: 45

Name: Bob, Age: 20, Student ID: 123456, Major: Computer Science

Name: John, Age: 25, Student ID: 654321, Major: Mathematics, Courses Teaching: Calculus Algebra

Name: Charlie, Age: 30, Student ID: 789012, Major: Physics, Research Area: Quantum Computing, Supervisor: Dr. Smith

**LAB2**

Imagine you're developing a university management system. You have a base class named Person that stores basic information about individuals associated with the university, such as:

name (string)

id (int)

Question:

Design a class hierarchy using inheritance to represent different types of people within the university. Consider the following categories:

Student: Inherits from Person and has additional attributes like:

major (string)

gpa (double)

A method calculateSemesterGPA(vector<double> grades) that takes a vector of grades (doubles) and calculates the semester GPA.

Faculty: Inherits from Person and has additional attributes like:

department (string)

title (string) - e.g., "Professor", "Lecturer"

A method teachCourse(string courseName) that simulates assigning a faculty member to teach a specific course.

Additional Considerations:

You can introduce further derived classes if you think of more specific roles within the university (e.g., Staff, Administrator).

Think about access specifiers (public, private, protected) for member variables and methods in the base and derived classes.

Consider virtual functions (especially in the context of polymorphism) if there's common functionality that might have different implementations in derived classes.

Guiding Tips:

Focus on code clarity and maintainability.

Use meaningful variable and method names.

Add comments to explain your design choices.

Test your code to ensure it works as expected.

#include <iostream>

#include <vector>

#include <string>

#include <numeric>

using namespace std;

class Person {

protected:

string name;

int id;

public:

Person(const string& name, int id) : name(name), id(id) {}

virtual void displayInfo() const {

cout << "Name: " << name << ", ID: " << id << endl;

}

virtual ~Person() {}

};

class Student : public Person {

private:

string major;

double gpa;

public:

Student(const string& name, int id, const string& major, double gpa)

: Person(name, id), major(major), gpa(gpa) {}

double calculateSemesterGPA(const vector<double>& grades) {

if (grades.empty()) return 0.0;

double sum = accumulate(grades.begin(), grades.end(), 0.0);

return sum / grades.size();

}

void displayInfo() const override {

Person::displayInfo();

cout << "Major: " << major << ", GPA: " << gpa << endl;

}

};

class Faculty : public Person {

private:

string department;

string title;

public:

Faculty(const string& name, int id, const string& department, const string& title)

: Person(name, id), department(department), title(title) {}

void teachCourse(const string& courseName) {

cout << title << " " << name << " is teaching " << courseName << " in the " << department << " department." << endl;

}

void displayInfo() const override {

Person::displayInfo();

cout << "Department: " << department << ", Title: " << title << endl;

}

};

int main() {

Student student("Alice", 1001, "Computer Science", 3.9);

student.displayInfo();

vector<double> grades = {3.5, 3.7, 4.0, 3.8};

cout << "Semester GPA: " << student.calculateSemesterGPA(grades) << endl;

Faculty faculty("Dr. Bob", 2001, "Mathematics", "Professor");

faculty.displayInfo();

faculty.teachCourse("Calculus");

return 0;

}

OUTPUT:

Name: Alice, ID: 1001

Major: Computer Science, GPA: 3.9

Semester GPA: 3.75

Name: Dr. Bob, ID: 2001

Department: Mathematics, Title: Professor

Professor Dr. Bob is teaching Calculus in the Mathematics department.