Day-3:

Base Class (Person):

1).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>

// Base class

class Person {

protected:

std::string name;

int age;

public:

Person(std::string n, int a) : name(n), age(a) {}

virtual void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

virtual ~Person() {} // Virtual destructor for proper cleanup

};

// Derived class

class Student : public Person {

public:

Student(std::string n, int a) : Person(n, a) {}

void getDetails() override {

std::cout << "Student Details -\n";

Person::getDetails();

}

};

int main() {

Student s("Ravi teja", 25);

s.getDetails();

return 0;

}

Output:

Student Details -

Name: Ravi teja

Age: 25

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(std::string n, int a) : name(n), age(a) {}

virtual void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

virtual ~Person() {}

};

class Student : public Person {

private:

int studentId;

std::string major;

public:

Student(std::string n, int a, int id, std::string m) : Person(n, a), studentId(id), major(m) {}

void setMajor(std::string m) {

major = m;

}

std::string getMajor() {

return major;

}

void getDetails() override {

std::cout << "Student Details -\n";

Person::getDetails();

std::cout << "Student ID: " << studentId << "\nMajor: " << major << std::endl;

}

};

class Faculty : public Person {

private:

std::string department;

public:

Faculty(std::string n, int a, std::string dept) : Person(n, a), department(dept) {}

void setDepartment(std::string dept) {

department = dept;

}

std::string getDepartment() {

return department;

}

void getDetails() override {

std::cout << "Faculty Details -\n";

Person::getDetails();

std::cout << "Department: " << department << std::endl;

}

};

int main() {

Student s("vasu", 24, 123, "Computer Science");

s.getDetails();

Faculty f("Dr. siva", 40, "English");

f.getDetails();

return 0;

}

Output:

Student Details -

Name: vasu

Age: 24

Student ID: 123

Major: Computer Science

Faculty Details -

Name: Dr. siva

Age: 40

Department: English

3). Inherits from Person

Data members: department (string), employeeId (int)

Member functions:

setDepartment(string) to set the faculty member's department

getDepartment() to retrieve the department

Override getDetails() to include faculty-specific information

Derived Class (TeachingAssistant): (Multilevel Inheritance)

#include <iostream>

#include <string

class Person {

protected:

std::string name;

int age;

public:

Person(std::string n, int a) : name(n), age(a) {}

virtual void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

virtual ~Person() {}

};

class Faculty : public Person {

protected:

std::string department;

int employeeId;

public:

Faculty(std::string n, int a, std::string dept, int id) : Person(n, a), department(dept), employeeId(id) {}

void setDepartment(std::string dept) {

department = dept;

}

std::string getDepartment() {

return department;

}

void getDetails() override {

std::cout << "Faculty Details -\n";

Person::getDetails();

std::cout << "Department: " << department << "\nEmployee ID: " << employeeId << std::endl;

}

};

class TeachingAssistant : public Faculty {

private:

std::string course;

public:

TeachingAssistant(std::string n, int a, std::string dept, int id, std::string course)

: Faculty(n, a, dept, id), course(course) {}

void setCourse(std::string c) {

course = c;

}

std::string getCourse() {

return course;

}

void getDetails() override {

std::cout << "Teaching Assistant Details -\n";

Faculty::getDetails();

std::cout << "Course: " << course << std::endl;

}

};

int main() {

TeachingAssistant ta("Emily Brown", 28, "Computer Science", 12345, "Introduction to Programming");

ta.getDetails();

return 0;

}

Output:

Teaching Assistant Details -

Faculty Details -

Name: Emily Brown

Age: 28

Department: Computer Science

Employee ID: 12345

Course: Introduction to Programming

4). 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)

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

Person(std::string n, int a) : name(n), age(a) {}

void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

};

class Student : public Person {

protected:

int studentId;

std::vector<std::string> courses;

public:

Student(std::string n, int a, int id) : Person(n, a), studentId(id) {}

void addCourse(std::string course) {

courses.push\_back(course);

}

void setCourses(std::vector<std::string> coursesList) {

courses = coursesList;

}

std::vector<std::string> getCourses() {

return courses;

}

void getStudentDetails() {

std::cout << "Student ID: " << studentId << "\nCourses Enrolled:\n";

for (const auto& course : courses) {

std::cout << "- " << course << std::endl;

}

}

};

class ResearchAssistant : public Student {

private:

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

public:

ResearchAssistant(std::string n, int a, int id) : Student(n, a, id) {}

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

coursesTeaching = courses;

}

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

return coursesTeaching;

}

void getDetails() {

std::cout << "Research Assistant Details -\n";

Person::getDetails();

getStudentDetails();

std::cout << "Courses Teaching:\n";

for (const auto& course : coursesTeaching) {

std::cout << "- " << course << std::endl;

}

}

};

int main() {

ResearchAssistant ra("Emily Brown", 25, 12345);

ra.addCourse("Computer Science");

ra.addCourse("Data Structures");

ra.setCoursesTeaching({"Introduction to Programming", "Algorithms"});

ra.getDetails();

return 0;

}

Output:

Research Assistant Details -

Name: Emily Brown

Age: 25

Student ID: 12345

Courses Enrolled:

- Computer Science

- Data Structures

Courses Teaching:

- Introduction to Programming

- Algorithms

5). Inherits from Person (separate inheritance from Student)

Data members: researchArea (string), supervisor (string)

Member functions:

setResearchArea(string) to set the research area

getResearchArea() to retrieve the research area

setSupervisor(string) to set the research supervisor

getSupervisor() to retrieve the supervisor

Override getDetails() to include RA-specific information

Derived Class (GraduateStudentTA): (Hybrid Inheritance)

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

Person(std::string n, int a) : name(n), age(a) {}

void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

};

class Student {

protected:

int studentId;

std::vector<std::string> courses;

public:

Student(int id) : studentId(id) {}

void addCourse(std::string course) {

courses.push\_back(course);

}

void setCourses(std::vector<std::string> coursesList) {

courses = coursesList;

}

std::vector<std::string> getCourses() {

return courses;

}

void getStudentDetails() {

std::cout << "Student ID: " << studentId << "\nCourses Enrolled:\n";

for (const auto& course : courses) {

std::cout << "- " << course << std::endl;

}

}

};

class GraduateStudentTA : public Person, public Student {

private:

std::string researchArea;

std::string supervisor;

public:

GraduateStudentTA(std::string n, int a, int id) : Person(n, a), Student(id) {}

void setResearchArea(std::string area) {

researchArea = area;

}

std::string getResearchArea() {

return researchArea;

}

void setSupervisor(std::string sup) {

supervisor = sup;

}

std::string getSupervisor() {

return supervisor;

}

void getDetails() {

std::cout << "Graduate Student TA Details -\n";

Person::getDetails();

getStudentDetails();

std::cout << "Research Area: " << researchArea << "\nSupervisor: " << supervisor << std::endl;

}

};

int main() {

GraduateStudentTA gst("Alice Johnson", 27, 12345);

gst.addCourse("Computer Science");

gst.addCourse("Data Structures");

gst.setResearchArea("Machine Learning");

gst.setSupervisor("Dr. Smith");

gst.getDetails();

return 0;

}

Output:

Graduate Student TA Details -

Name: Alice Johnson

Age: 27

Student ID: 12345

Courses Enrolled:

- Computer Science

- Data Structures

Research Area: Machine Learning

Supervisor: Dr. Smith

6). Inherits from both Student and TeachingAssistant (combines functionality)

Might have additional data members or functions specific to graduate student Tas

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

Person(std::string n, int a) : name(n), age(a) {}

void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

};

class Student : public Person {

protected:

int studentId;

std::vector<std::string> courses;

public:

Student(std::string n, int a, int id) : Person(n, a), studentId(id) {}

void addCourse(std::string course) {

courses.push\_back(course);

}

void setCourses(std::vector<std::string> coursesList) {

courses = coursesList;

}

std::vector<std::string> getCourses() {

return courses;

}

void getStudentDetails() {

std::cout << "Student ID: " << studentId << "\nCourses Enrolled:\n";

for (const auto& course : courses) {

std::cout << "- " << course << std::endl;

}

}

};

class TeachingAssistant : public Person {

protected:

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

public:

TeachingAssistant(std::string n, int a) : Person(n, a) {}

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

coursesTeaching = courses;

}

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

return coursesTeaching;

}

void getTeachingDetails() {

std::cout << "Courses Teaching:\n";

for (const auto& course : coursesTeaching) {

std::cout << "- " << course << std::endl;

}

}

}

class GraduateStudentTA : public Student, public TeachingAssistant {

private:

std::string researchArea;

std::string supervisor;

public:

GraduateStudentTA(std::string n, int a, int id)

: Person(n, a), Student(n, a, id), TeachingAssistant(n, a) {}

void setResearchArea(std::string area) {

researchArea = area;

}

std::string getResearchArea() {

return researchArea;

}

void setSupervisor(std::string sup) {

supervisor = sup;

}

std::string getSupervisor() {

return supervisor;

}

void getDetails() {

std::cout << "Graduate Student TA Details -\n";

Person::getDetails();

getStudentDetails();

getTeachingDetails();

std::cout << "Research Area: " << researchArea << "\nSupervisor: " << supervisor << std::endl;

}

};

int main() {

GraduateStudentTA gst(“ Johnson", 23, 9876);

gst.addCourse("Computer Science");

gst.addCourse("Data Structures");

gst.setCoursesTeaching({"Introduction to Programming", "Algorithms"});

gst.setResearchArea("Machine Learning");

gst.setSupervisor("Dr. siva");

gst.getDetails();

return 0;

}

Output:

Graduate Student TA Details -

Name: Johnson

Age: 23

Student ID: 9876

Courses Enrolled:

- Computer Science

- Data Structures

Courses Teaching:

- Introduction to Programming

- Algorithms

Research Area: Machine Learning

Supervisor: Dr. Siva