# TASK 1:

#include<iostream>

using namespace std;

class Time

{

public:

Time(){}

Time(int hr, int min)

{

this->hr = hr;

this->min = min;

}

void setTime(int hr, int min)

{

this->hr = hr;

this->min = min;

}

void getTime(int& hr, int& min)

{

cout << hr << ":" << min << endl;

}

void printTime()

{

cout << hr << ":" << min << endl;

}

private:

int hr;

int min;

};

class Date

{

public:

Date(){}

Date(int day, int month, int year)

{

this-> day= day;

this->month = month;

this->year = year;

}

void setDate(int day, int month, int year)

{

this->day = day;

this->month = month;

this->year = year;

}

void getDate(int& day, int& month, int& year)

{

cout << day << "/" << month << "/" << year << endl;

}

void printDate()

{

cout << day << "/" << month << "/" << year << endl;

}

private:

int month;

int day;

int year;

};

class Event

{

public:

Event(int hours = 0, int minutes = 0, int m = 1, int d = 1, int y = 1900, string name = "Christmas")

{

this->eventTime.setTime(05, 00);

this->eventDay.setDate(25, 8, 2022);

this->eventName = name;

}

void setEventData(int hours, int minutes, int m, int d, int y, string name)

{

this->eventTime.setTime(hours, minutes);

this->eventDay.setDate(d, m, y);

this->eventName = name;

}

void printEventData()

{

cout << "Event Name is: " << eventName << endl;

cout << "Event Day is: " << endl;

eventDay.printDate();

cout << "Event Time is: " << endl;

eventTime.printTime();

}

private:

string eventName;

Time eventTime;

Date eventDay;

};

int main()

{

Event e;

e.setEventData(12, 30, 7, 4, 2022, "EID");

e.printEventData();

}

# OUTPUT:

# 

# TASK 2:

#include<iostream>

using namespace std;

class Car

{

public:

private:

};

class engine

{

public:

private:

};

class wheel

{

public:

private:

};

class window

{

public:

private:

};

class door

{

public:

private:

};

class tires

{

public:

private:

};

int main()

{}

# OUTPUT:

# TASK 3:

#include <iostream>

#include<string.h>

using namespace std;

class Address

{

public:

int house;

string city, country;

Address(int house, string city, string country)

{

this->house = house;

this->city = city;

this->country = country;

}

int housegetter()

{

return house;

}

string citygetter()

{

return city;

}

string countrygetter()

{

return country;

}

};

class Person

{

private:

Address\* address;

public:

string name;

Person(string name, Address\* address)

{

this->name = name;

this->address = address;

}

void display()

{

cout << "Name of Person is: " << name << endl;

cout << "Address: " << endl << "House: " << address->house << " City: " << address->city << " Country: " << address->country << endl << endl;

}

};

int main()

{

Address add1 = Address(55, "Lahore", "Pakistan");

Person p1 = Person("Huzaifa", &add1);

Person p2 = Person("Ali", &add1);

Person p3 = Person("Ahmad", &add1);

p1.display();

p2.display();

p3.display();

}

# OUTPUT:

# 

# TASK 4:

#include <iostream>

#include <string>

using namespace std;

class Person

{

string name;

public:

Person() {}

Person(string name)

{

this->name;

}

string getName()

{

return name;

}

};

class Teacher

{

Person p;

public:

Teacher() {}

Teacher(Person p)

{

this->p = p;

}

};

int main()

{

Person person{ "Ali" }; // Create a Person outside the scope of Teacer

{

Teacher t{ person }; // Create a Teacher and use the constructor parameter to pass the Person to it.

}

// Teacher goes out of scope here and is destroyed.

// Ali still exists here, but the Teacher doesn't

cout << person.getName() << " still exists!";

}

# OUTPUT:

# TASK 5:

**COMPOSITION:**

1. **School and Classroom:**

A school can contain multiple classrooms. If we delete the school, the classrooms will automatically be deleted. So Child objects cannot live without the Parent object. If a parent object is deleted, all its child objects will also be deleted.

1. **Pages and Book:**

A Book can contain multiple classrooms. If we delete the Book, the classrooms will automatically be deleted. So Child objects cannot live without the Parent object. If a parent object is deleted, all its child objects will also be deleted.

1. **Phone and CPU:**

A Phone can contain multiple classrooms. If we delete the Phone, the classrooms will automatically be deleted. So Child objects cannot live without the Parent object. If a parent object is deleted, all its child objects will also be deleted.

1. **Car and Engine:**

A CAR can contain multiple classrooms. If we delete the Car, the classrooms will automatically be deleted. So Child objects cannot live without the Parent object. If a parent object is deleted, all its child objects will also be deleted.

1. **Lab and Computers:**

A LAB can contain multiple classrooms. If we delete the LAB, the classrooms will automatically be deleted. So Child objects cannot live without the Parent object. If a parent object is deleted, all its child objects will also be deleted.

**AGGREGATION:**

1. **Teacher - Department.**

A Teacher may belong to multiple departments. But if we delete a Department, the Teacher will still be there. So here both objects have their own life cycle, but there exist an ownership.

1. **Person and Address:**

A Person may belong to multiple Addresses. But if we delete a Addresses, the Person will still be there. So here both objects have their own life cycle, but there exist an ownership.

1. **Book and Pen:**

A Book may belong to multiple pens. But if we delete a Pen, the Book will still be there. So here both objects have their own life cycle, but there exist an ownership.

1. **Person and Phone:**

Person may belong to multiple Phones. But if we delete a Phone, the Person will still be there. So here both objects have their own life cycle, but there exist an ownership.

1. **Laptop and Mouse:**

Laptop may belong to multiple Mouse. But if we delete a Mouse, the Laptop will still be there. So here both objects have their own life cycle, but there exist an ownership.