Name Jawad Ahmed

Roll No : K24-0678

**Assignment #2**

**Question 1:**

**Source Code:**

#include <iostream>

using **namespace** std;

**class** AssignmentInfo

{

    string name;

    string id;

**public:**

    AssignmentInfo()

    {

        name = "Jawad Ahmed";

        id = "K24-0678";

    }

**void** WhoDidAssignment()

    {

        cout << "\n\nThe assignment was created by " << name << " (" << id << ")\n"

             << endl;

    }

};

**int** getValidInt()

{

**int** value;

    while (true)

    {

        cout << "Enter your choice: ";

        cin >> value;

        if (cin.fail())

        {

            cin.clear(); *// Clear the error flag*

            cin.ignore(1000, '\n'); *// Discard invalid input*

            cout << "Invalid input! Please enter a valid number.\n";

        }

        else

        {

            cin.ignore(1000, '\n'); *// Clear any extra characters (like letters after numbers)*

            return value; *// Return valid input*

        }

    }

}

**class** Bus;

**class** Customer

{

**protected:**

**int** CustomerId;

    string Name;

**bool** ispaid;

    string pickup;

    string drop;

**int** CardActiveMonths;

    Bus \*AssignedBus;

**public:**

    Customer()

    {

        CustomerId = 0;

        Name = "";

        ispaid = false;

        pickup = "";

        drop = "";

        this->AssignedBus = nullptr;

    }

    Customer(**int** id, string name, **bool** paid, string pickup, string drop, **int** monthsPaid)

    {

        CustomerId = id;

        Name = name;

        ispaid = paid;

        this->pickup = pickup;

        this->drop = drop;

        this->AssignedBus = nullptr;

        CardActiveMonths = monthsPaid;

    }

**virtual** **void** printDetails()

    {

        cout << "Is Paid: " << ispaid << endl;

        cout << "Pickup: " << pickup << endl;

        cout << "Drop: " << drop << endl;

    }

**int** getCustomerId()

    {

        return CustomerId;

    }

    string getName()

    {

        return Name;

    }

**void** setName(string name)

    {

        this->Name = name;

    }

**bool** getIsPaid()

    {

        return ispaid;

    }

    string getPickup()

    {

        return pickup;

    }

    string getDrop()

    {

        return drop;

    }

**virtual** **void** setIsPaid(**bool** paid)

    {

        ispaid = paid;

    }

**void** setPickup(string pickup)

    {

        this->pickup = pickup;

    }

**void** setDrop(string drop)

    {

        this->drop = drop;

    }

    Bus **\***getAssignedBus();

**void** setAssignedBus(Bus **\***bus)

    {

        this->AssignedBus = bus;

    }

};

**class** Student : **public** Customer

{

**public:**

    Student(**int** id, string name, **bool** ispaid, string pickup, string drop) : Customer(id, name, ispaid, pickup, drop, 6) {}

**void** setIsPaid(**bool** paid) **override**

    {

        ispaid = paid;

        CardActiveMonths = 6;

    }

**void** printDetails() **override**

    {

        cout << "\nStudent ID: " << CustomerId << endl;

        cout << "Student Name: " << Name << endl;

        Customer::printDetails();

        cout << "Card Active Months: "

             << CardActiveMonths << endl

             << endl;

    }

};

**class** Teacher : **public** Customer

{

**public:**

    Teacher(**int** id, string name, **bool** ispaid, string pickup, string drop) : Customer(id, name, ispaid, pickup, drop, 1) {}

**void** setIsPaid(**bool** paid) **override**

    {

        ispaid = paid;

        CardActiveMonths = 1;

    }

**void** printDetails() **override**

    {

        cout << "\nTeacher ID: " << CustomerId << endl;

        cout << "Teacher Name: " << Name << endl;

        Customer::printDetails();

        cout << "Card Active Months: \n"

             << CardActiveMonths << endl;

    }

};

**class** Bus

{

**int** BusId;

**int** capacity;

    string \*stops;

**int** NoCustomerAssigned;

**int** noStops;

**public:**

    Bus(**int** id, **int** cap, string **\***stop, **int** noStops)

    {

        BusId = id;

        capacity = cap;

        this->noStops = noStops;

        stops = new string[noStops];

        for (**int** i = 0; i < noStops; i++)

        {

            stops[i] = stop[i];

        }

        NoCustomerAssigned = 0;

    }

    Bus()

    {

        BusId = 0;

        capacity = 0;

        stops = nullptr;

        NoCustomerAssigned = 0;

        noStops = 0;

    }

    Bus(**const** Bus **&**other)

    {

        cout << "\nBus ke copy constructor me";

        BusId = other.BusId;

        capacity = other.capacity;

        noStops = other.noStops;

        stops = new string[noStops];

        NoCustomerAssigned = other.NoCustomerAssigned;

        for (**int** i = 0; i < noStops; i++)

        {

            stops[i] = other.stops[i];

        }

    }

*// Bus &operator=(const Bus &other)*

*// {*

*//     cout << "Assignment operator called for Bus: " << other.BusId << endl;*

*//     if (this != &other)*

*//     {*

*//         delete[] stops;*

*//         BusId = other.BusId;*

*//         capacity = other.capacity;*

*//         noStops = other.noStops;*

*//         NoCustomerAssigned = other.NoCustomerAssigned;*

*//         stops = new string[noStops];*

*//         for (int i = 0; i < noStops; i++)*

*//         {*

*//             stops[i] = other.stops[i];*

*//         }*

*//     }*

*//     return \*this;*

*// }*

    ~Bus()

    {

        delete[] stops;

    }

**int** getBusId()

    {

        return BusId;

    }

**int** getCapacity()

    {

        return capacity;

    }

    string **\***getStops()

    {

        return stops;

    }

**void** setNoStops(**int** stops)

    {

        this->noStops = stops;

    }

**int** getNoStops()

    {

        return noStops;

    }

**void** setStops(string **\***stop, **int** noStops)

    {

        delete[] stops;

        this->noStops = noStops;

        stops = new string[noStops];

        for (**int** i = 0; i < noStops; i++)

        {

            stops[i] = stop[i];

        }

    }

**void** setNoCustomersAssigned(**int** noCustomersAssigned)

    {

        this->NoCustomerAssigned = noCustomersAssigned;

    }

**int** getNoCustomersAssigned()

    {

        return NoCustomerAssigned;

    }

**void** setCapacity(**int** capacity)

    {

        this->capacity = capacity;

    }

**void** setBusId(**int** busId)

    {

        this->BusId = busId;

    }

**void** printStops()

    {

        for (**int** i = 0; i < noStops; i++)

        {

            cout << "\n"

                 << i + 1 << ". " << stops[i];

        }

        cout << endl;

    }

**void** AssignCustomer(Customer **\***Customer)

    {

        if (NoCustomerAssigned < capacity)

        {

**bool** pickupFound = false;

**bool** dropFound = false;

            for (**int** i = 0; i < noStops; i++)

            {

                if (Customer->getPickup() == stops[i])

                {

                    pickupFound = true;

                    break;

                }

            }

            if (pickupFound)

            {

                for (**int** j = 0; j < noStops; j++)

                {

                    if (Customer->getDrop() == stops[j])

                    {

                        dropFound = true;

                        break;

                    }

                }

            }

            if (pickupFound && dropFound)

            {

                Customer->setIsPaid(true);

                NoCustomerAssigned++;

                Customer->setAssignedBus(this);

                cout << "Customer " << Customer->getName() << " has been assigned to bus " << BusId << endl;

            }

            else

            {

                cout << "Customer stop doesn't exist" << endl;

            }

        }

        else

        {

            cout << "Bus is full" << endl;

        }

    }

};

**class** TransportSystem

{

    Customer \*\*customer;

    Bus \*bus;

**int** noCustomers;

**int** noBuses;

**public:**

    Customer **\***getCustomer()

    {

        return \*customer;

    }

    Bus **\***getBus()

    {

        return bus;

    }

**void** setBus(Bus **\***bus)

    {

        this->bus = bus;

    }

**void** SetCustomer(Customer **\*\***customer)

    {

        this->customer = customer;

    }

    TransportSystem()

    {

        noCustomers = 0;

        noBuses = 0;

        customer = nullptr;

        bus = nullptr;

    }

**void** printAllCustomers()

    {

        if (noCustomers == 0)

        {

            cout << "No Customers found\n"

                 << endl;

            return;

        }

        cout << "\nAll Customers: \n";

        for (**int** i = 0; i < noCustomers; i++)

        {

            customer[i]->printDetails();

        }

    }

**void** printAllBuses()

    {

        if (noBuses == 0)

        {

            cout << "No buses found\n"

                 << endl;

            return;

        }

        cout << "\nAll Buses: \n";

        for (**int** i = 0; i < noBuses; i++)

        {

            cout << "Bus ID: " << bus[i].getBusId() << ", Capacity: " << bus[i].getCapacity() << ", No of Customers: " << bus[i].getNoCustomersAssigned() << ", Stops: ";

            bus[i].printStops();

        }

    }

**void** addCustomer(string name, **bool** paid, string pickup, string drop, string Type)

    {

        Customer \*\*temp = new Customer \*[noCustomers + 1]; *// ✅ Allocate array of pointers*

        for (**int** i = 0; i < noCustomers; i++)

        {

            temp[i] = customer[i]; *// ✅ Copy existing pointers*

        }

        if (Type == "student")

        {

            temp[noCustomers] = new Student(noCustomers + 1, name, paid, pickup, drop); *// ✅ Use new*

        }

        else if (Type == "teacher")

        {

            temp[noCustomers] = new Teacher(noCustomers + 1, name, paid, pickup, drop); *// ✅ Use new*

        }

        delete[] customer; *// ✅ Free old array (but don't delete objects)*

        customer = temp;

        noCustomers++;

    }

**void** addBus(**int** id, **int** cap, string **\***stop, **int** noStops)

    {

        Bus \*temp = new Bus[noBuses + 1];

        cout << "No Busses in Tranport : " << noBuses << endl;

        for (**int** i = 0; i < noBuses; i++)

        {

            temp[i] = bus[i];

        }

        temp[noBuses] = Bus(id, cap, stop, noStops);

        delete[] bus;

        bus = temp;

        noBuses++;

    }

};

Bus **\***Customer::getAssignedBus()

{

    if (AssignedBus)

    {

        cout << "Assigned Bus: " << AssignedBus->getBusId() << endl;

        return AssignedBus;

    }

    else

        cout << "No bus assigned yet." << endl;

    return nullptr;

}

**int** main()

{

    AssignmentInfo assignment;

    assignment.WhoDidAssignment();

    TransportSystem ts;

**int** choice;

    string name;

**int** capacity;

    string pickup;

    string drop;

    while (true)

    {

        cout << "Main Menu" << endl;

        cout << "1. Add Customer" << endl;

        cout << "2. Add Bus" << endl;

        cout << "3. Assign Bus" << endl;

        cout << "4. All Customers" << endl;

        cout << "5. All Buses" << endl;

        cout << "6. Exit" << endl;

        choice = getValidInt();

        switch (choice)

        {

        case 1:

        {

            cout << "Enter name: ";

            cin >> name;

            cout << "Enter pickup location: ";

            cin >> pickup;

            cout << "Enter drop location: ";

            cin >> drop;

            cout << "Student / Teacher (lowercase)";

            string Type;

            cin >> Type;

            ts.addCustomer(name, false, pickup, drop, Type);

        }

        break;

        case 2:

        {

            cout << "Enter bus ID: (integer)";

**int** busId;

            cin >> busId;

            cout << "Enter capacity: (integer)";

            cin >> capacity;

            cout << "Enter number of stops: (Integer greater than 0)";

**int** n;

            cin >> n;

            string \*stops = new string[n];

            for (**int** i = 0; i < n; i++)

            {

                cout << "Enter stop: ";

                cin >> stops[i];

            }

            ts.addBus(busId, capacity, stops, n);

        }

        break;

        case 3:

        {

            cout << "Enter Customer index: ";

**int** CustomerIndex;

            cin >> CustomerIndex;

            cout << "Enter bus index: ";

**int** busIndex;

            cin >> busIndex;

            ts.getBus()[busIndex].AssignCustomer(&ts.getCustomer()[CustomerIndex]);

        }

        break;

        case 4:

            ts.printAllCustomers();

            break;

        case 5:

            ts.printAllBuses();

            break;

        case 6:

            return 0;

        default:

            cout << "Invalid choice. Try again." << endl;

        }

    }

    return 0;

}

**Class Diagram:**

|  |
| --- |
| **BUS** |
| **- BusId: int**  **- capacity: int**  **- stops: string\***  **- NoStudentAssigned: int**  **- noStops: int** |
| **+ Bus()**  **+ Bus(id, cap, stop, noStops)**  **+ Bus(const Bus&)**  **+ ~Bus() + getBusId(): int**  **+ getCapacity(): int**  **+ getStops(): string\***  **+ setNoStops(int)**  **+ getNoStops(): int**  **+ setStops(string\*, int)**  **+ getNoStudentsAssigned(): int + setCapacity(int) + setBusId(int) + printStops()**  **+ AssignStudent(Student\*)** |

M

1

M

|  |
| --- |
| **CUSTOMER** |
| **#StudentId**  **#Name**  **#pickup**  **#drop**  **#bus** |
| **+ Student()**  **+ Student(id, name, paid, pickup, drop)**  **+ getStudentId(): int**  **+ getName(): string**  **+ setName(string)**  **+ getIsPaid(): bool**  **+ setIsPaid(bool)**  **+ getPickup(): string**  **+ getDrop(): string**  **+ setPickup(string)**  **+ setDrop(string)**  **+ getAssignedBus(): Bus\***  **+ setAssignedBus(Bus\*)** |

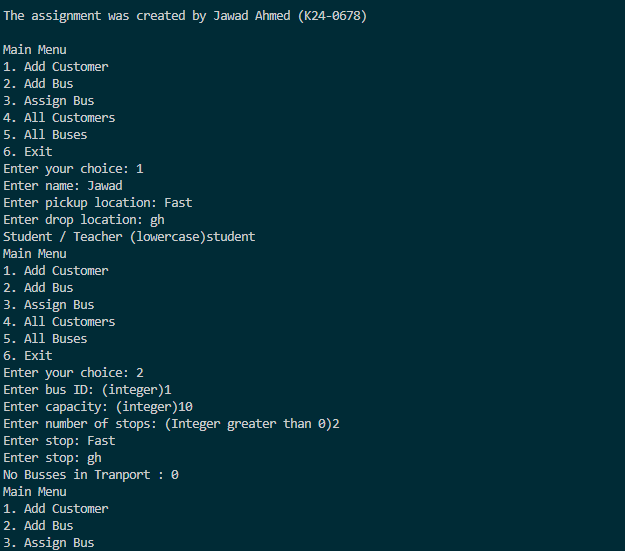
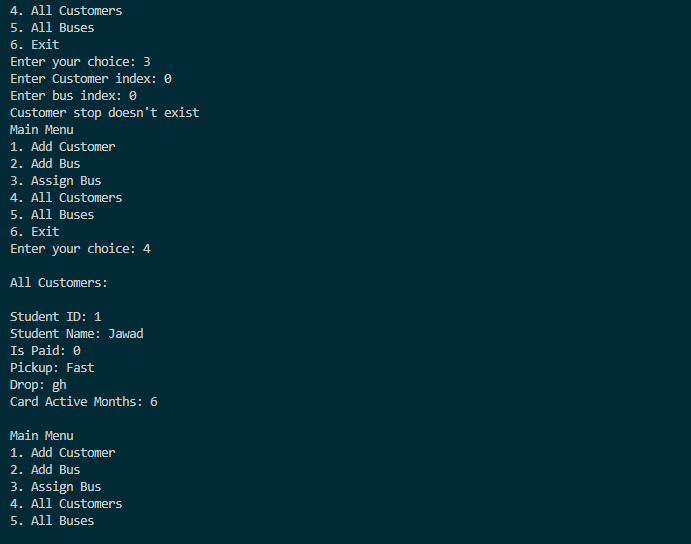
|  |
| --- |
| **Student** |
| **//Attributes from Customer** |
| **+setIsPaid()//override**  **+printDetails()/override** |

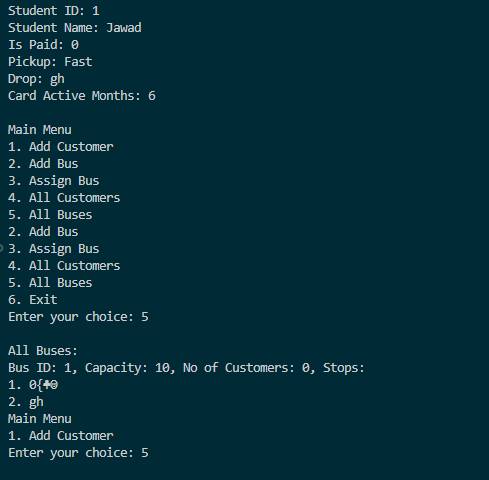
1

|  |
| --- |
| **Transport System** |
| **- Customer: customer\***  **- bus: Bus\***  **- noStudents: int**  **- noBuses: int** |
| **+ TransportSystem()**  **+ printAllStudents()**  **+ printAllBuses()**  **+ addStudent()**  **+ addBus()**  **+ getStudent(): Student\***  **+ getBus(): Bus\***  **+ setBus(Bus\*)**  **+ setStudent(Student\*)** |

|  |
| --- |
| **Teacher** |
| **//Attributes from Customer** |
| **+setIsPaid()//override**  **+printDetails()/override** |

**Output:**

**** ****

****

**Question 2:**

**Source Code:**

#include <iostream>

using **namespace** std;

**class** AssignmentInfo

{

    string name;

    string id;

**public:**

    AssignmentInfo()

    {

        name = "Jawad Ahmed";

        id = "K24-0678";

    }

**void** WhoDidAssignment()

    {

        cout << "\n\nThe assignment was created by " << name << " (" << id << ")\n"

             << endl;

    }

};

**class** Ghost

{

**protected:**

**int** RNG;

    string nameWorker;

**int** ScareLevel;

**public:**

    Ghost()

    {

        RNG = 0;

        nameWorker = "";

        ScareLevel = 0;

    }

**int** getRNG() **const**

    {

        return RNG;

    }

    string getName() **const**

    {

        return nameWorker;

    }

**int** getScarelevel()

    {

        return ScareLevel;

    }

    Ghost(**int** rng, string name)

    {

        RNG = rng;

        nameWorker = name;

        ScareLevel = 0;

    }

**virtual** **void** Haunting() {};

**virtual** ~Ghost() {};

**friend** ostream **&**operator<<(ostream **&**os, **const** Ghost **&**ghost);

**virtual** Ghost **\***operator+(Ghost **&**ghost) = 0;

};

ostream **&**operator<<(ostream **&**os, **const** Ghost **&**ghost)

{

    os << "Ghost: " << ghost.nameWorker << ", RNG: " << ghost.RNG << ", Scare level: " << ghost.ScareLevel << endl;

    return os;

}

*// Ghost operator+(Ghost &ghost)*

*// {*

*//     return Ghost(ghost.RNG\*2,ghost.nameWorker);*

*// }*

**class** Poltergiests : **virtual** **public** Ghost

{

**public:**

    Poltergiests(**int** rng, string name) : Ghost(rng, name)

    {

        ScareLevel = 4;

    }

**void** Haunting() **override**

    {

        cout << nameWorker << "(Poltergeist)  haunts by moving objects. Scare level: " << ScareLevel << endl;

    }

    Ghost **\***operator+(Ghost **&**ghost) **override**

    {

        return new Poltergiests(ghost.getRNG(), ghost.getName());

    }

};

**class** Banshees : **virtual** **public** Ghost

{

**public:**

    Banshees(**int** rng, string name) : Ghost(rng, name)

    {

        ScareLevel = 6;

    }

**void** Haunting() **override**

    {

        cout << nameWorker << "(Banshee)  haunts by making loud noises. Scare level: " << ScareLevel << endl;

    }

    Ghost **\***operator+(Ghost **&**ghost) **override**

    {

        cout << "Ghost in Banshees +";

        Banshees \*tempghost = new Banshees(\*this); *// Create copy of current object*

        tempghost->RNG += ghost.getRNG(); *// Sum RNG values*

        tempghost->nameWorker += "+" + ghost.getName(); *// Combine names*

        tempghost->ScareLevel += ghost.getScarelevel(); *// Sum scare levels*

        return tempghost;

    }

*// In Banshees class*

    Banshees operator+()

    {

        cout << "Banshees +";

        return Banshees(RNG \* 2, "Boosted\_" + nameWorker); *// Doubles RNG*

    }

};

**class** ShadowGhosts : **virtual** **public** Ghost

{

**public:**

    ShadowGhosts(**int** rng, string name) : Ghost(rng, name)

    {

        ScareLevel = 9;

    }

**void** Haunting() **override**

    {

        cout << nameWorker << "(Shadow)  haunts by whisper creeping. Scare level: " << ScareLevel << endl;

    }

    Ghost **\***operator+(Ghost **&**ghost) **override**

    {

        cout << "Ghost in Banshees +";

        ShadowGhosts \*tempghost = new ShadowGhosts(\*this); *// Create copy of current object*

        tempghost->RNG = 2; *// Sum RNG values*

        tempghost->nameWorker += "+" + ghost.getName(); *// Combine names*

        tempghost->ScareLevel += ghost.getScarelevel(); *// Sum scare levels*

        return tempghost;

    }

};

**class** HybridGhosts : **virtual** **public** ShadowGhosts, **virtual** **public** Poltergiests

{

**public:**

    HybridGhosts(**int** rng, string name) : Ghost(rng, name), ShadowGhosts(rng, name), Poltergiests(rng, name) {};

**void** Haunting() **override**

    {

        cout << nameWorker << "(Hybrid)  haunts by moving objects, making loud noises, and whisper creeping. Scare level: " << ScareLevel << endl;

*// cout << nameWorker << "(Shadow)  haunts by whisper creeping. Scare level: " << ScareLevel << endl;*

    }

    Ghost **\***operator+(Ghost **&**ghost) **override**

    {

        return new HybridGhosts(ghost.getRNG(), ghost.getName());

    }

};

**class** HauntedHouse

{

    Ghost \*\*ghost;

**int** ghostLength;

**public:**

*// Poltergiests p1;*

*// Banshees b1;*

*// ShadowGhosts s1;*

*// HybridGhosts h1;*

*// public:*

    Ghost **\*\***getGhost()

    {

        return ghost;

    }

**int** getLengthGhost()

    {

        return ghostLength;

    }

    HauntedHouse()

    {

        ghostLength = 0;

        ghost = new Ghost \*[ghostLength];

    };

**void** addGhost(Ghost **\***g)

    {

        Ghost \*\*temp = new Ghost \*[ghostLength + 1];

        for (**int** i = 0; i < ghostLength; i++)

            temp[i] = ghost[i];

        temp[ghostLength] = g;

        ghostLength++;

        delete[] ghost;

        ghost = temp;

    }

*// Poltergiests getPolt(){*

*//     return p1;*

*// }*

*// Banshees getBan(){*

*//     return b1;*

*// }*

*// ShadowGhosts getSha(){*

*//     return s1;*

*// }*

*// HybridGhosts getHyp(){*

*//     return h1;*

*// }*

**void** StartHaunting()

    {

*// p1.Haunting();*

*// b1.Haunting();*

*// s1.Haunting();*

*// h1.Haunting();*

        for (**int** i = 0; i < ghostLength; i++)

        {

            ghost[i]->Haunting();

*// Visits();*

        }

    };

**void** **friend** Visits()

    {

    }

};

**class** Visitors

{

    string name;

**int** BraveryScale;

    string Brave;

**public:**

    string getName()

    {

        return name;

    }

**int** getBravery()

    {

        return BraveryScale;

    }

    Visitors(string name, **int** bravery)

    {

        this->name = name;

        this->BraveryScale = bravery;

        if (BraveryScale >= 8 && BraveryScale <= 10)

        {

            Brave = "Fearless";

        }

        else if (BraveryScale >= 5 && BraveryScale < 8)

        {

            Brave = "Brave";

        }

        else if (BraveryScale >= 1 && BraveryScale < 5)

        {

            Brave = "Cautious";

        }

    }

};

**void** Visits(Visitors **\***v, HauntedHouse h, **int** VLength)

{

*// Ghost \*\*ptr = h.getGhost();*

    for (**int** i = 0; i < VLength; i++)

    {

        cout << "Visitor: " << v[i].getName() << ", Bravery: " << v[i].getBravery() << endl;

        for (**int** j = 0; j < h.getLengthGhost(); j++)

        {

            cout << h.getGhost()[j]->getName() << endl;

            if (v[i].getBravery() > h.getGhost()[j]->getRNG())

            {

                cout << "Laughs" << endl;

            }

            else if (v[i].getBravery() < h.getGhost()[j]->getRNG())

            {

                cout << "Screams and runs away" << endl;

            }

            else

            {

                cout << "Gets a Shaky Voice" << endl;

            }

        }

    }

};

**int** main()

{

    AssignmentInfo a;

    a.WhoDidAssignment();

    Visitors v1("Tom", 8);

    Visitors v2("Mike", 6);

    Visitors VisitorArray[2] = {v1, v2};

    Poltergiests p1(3, "Tom");

    Banshees b1(5, "Mike");

    ShadowGhosts s1(10, "Tom");

    HybridGhosts h1(10, "Mike");

    HauntedHouse h;

    h.addGhost(&p1);

    h.addGhost(&h1);

    Visits(VisitorArray, h, 2);

    h.StartHaunting();

    cout << b1;

*// Banshees bs(4,"BansheeTest")*

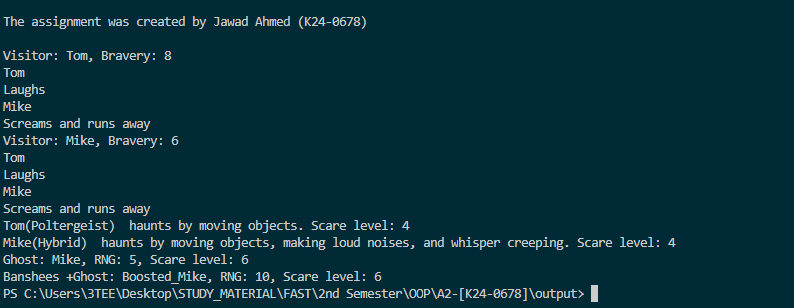
    Banshees bs = +b1;

    cout << bs;

    return 0;

}

**Output:**

****

**Question 3:**

**Source Code:**

#include <iostream>

#include <math.h>

using **namespace** std;

**class** AssignmentInfo

{

    string name;

    string id;

**public:**

    AssignmentInfo()

    {

        name = "Jawad Ahmed";

        id = "K24-0678";

    }

**void** WhoDidAssignment()

    {

        cout << "\n\nThe assignment was created by " << name << " (" << id << ")\n"

             << endl;

    }

};

**class** ConflictResolution;

**class** System;

**class** vehicle

{

**protected:**

**int** speed;

**int** capacity;

**int** energyEfficiency;

**int** vehicleId;

**int** **static** NoDeliviries;

**public:**

    vehicle(**int** speed, **int** capacity, **int** energyEfficiency, **int** id)

    {

        this->speed = speed;

        this->capacity = capacity;

        this->energyEfficiency = energyEfficiency;

        this->vehicleId = id;

    }

**void** DeliveryRoute()

    {

    }

**virtual** **double** CalcDistance(**int** x1, **int** y1, **int** x2, **int** y2)

    {

**int** distance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

        return distance;

    }

**virtual** **double** CalTime(**double** distance, **int** penaltyItems)

    {

**double** time = distance / speed;

        time += penaltyItems \* 0;

        cout << "Time: " << time << " hours" << endl;

        return time;

    }

**friend** **class** ConflictResolution;

**friend** **class** System;

};

**class** RamzanDrone : **public** vehicle

{

**public:**

    RamzanDrone(**int** id) : vehicle(4, 1, 9, id) {};

**double** CalcDistance(**int** x1, **int** y1, **int** x2, **int** y2) **override**

    {

**int** distance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

        cout << "Ramzan Drone Distance: " << distance << " km" << endl;

        return distance;

    }

**double** CalTime(**double** distance, **int** penaltyItems) **override**

    {

**double** time = distance / speed;

        time += penaltyItems \* 0.1;

        cout << "Time: " << time << " hours" << endl;

        return time;

    }

};

**class** RamzanTimeShip : **public** vehicle

{

**public:**

    RamzanTimeShip(**int** id) : vehicle(10, 10, 1, id) {};

**double** CalcDistance(**int** x1, **int** y1, **int** x2, **int** y2) **override**

    {

**int** SpaceDistance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

**double** penalty = 10;

        return SpaceDistance + penalty;

        cout << "Ramzan Time Ship Distance: " << SpaceDistance + penalty << " km" << endl;

    }

**double** CalTime(**double** distance, **int** penaltyItems) **override**

    {

**double** time = distance / speed;

        time += penaltyItems \* 0.5;

        cout << "Time: " << time << " hours" << endl;

        return time;

    }

};

**class** RamzanHyperPod : **public** vehicle

{

**public:**

    RamzanHyperPod(**int** id) : vehicle(20, 20, 2, id) {};

**double** CalcDistance(**int** x1, **int** y1, **int** x2, **int** y2) **override**

    {

**double** euclideanDist = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

        return euclideanDist + (rand() % 10);

    }

**double** CalTime(**double** distance, **int** penaltyItems) **override**

    {

**double** time = distance / speed;

        time += penaltyItems \* 0;

        cout << "Time: " << time << " hours" << endl;

        return time;

    }

};

**class** Package

{

**public:**

**int** weight;

    string urgency;

**int** Destx;

**int** Desty;

    Package(**int** weight, string urgency, **int** x, **int** y)

    {

        this->weight = weight;

        this->urgency = urgency;

        this->Destx = x;

        this->Desty = y;

    }

};

**class** ConflictResolution

{

**public:**

**static** vehicle **&**resolveConflict(vehicle **&**v1, vehicle **&**v2, Package **\***p)

    {

**double** distance1 = v1.CalcDistance(0, 0, p->Destx, p->Desty);

**double** distance2 = v2.CalcDistance(0, 0, p->Destx, p->Desty);

**double** time1 = v1.CalTime(distance1, p->weight);

**double** time2 = v2.CalTime(distance2, p->weight);

*// Scoring System: Lower time and higher speed are prioritized*

**double** score1 = (time1 \* 2) + (v1.speed \* -1) + (v1.capacity \* -0.5) + (v1.energyEfficiency \* -0.2);

**double** score2 = (time2 \* 2) + (v2.speed \* -1) + (v2.capacity \* -0.5) + (v2.energyEfficiency \* -0.2);

        return (score1 < score2) ? v1 : v2;

    }

};

**class** System

{

    Package \*\*p;

**int** pLength;

**public:**

    System(Package **\*\***p, **int** pLength)

    {

        this->p = p;

        this->pLength = pLength;

    }

**void** command(string command, **int** index)

    {

        if (command == "Deliver")

        {

            vehicle \*v = AssignVehicle(p[index]);

**int** vId = v->vehicleId;

            cout << "Assigned vehicle: " << vId << endl;

            if (vId >= 100 and vId < 200)

            {

                cout << "Ramzan Drone is being used." << endl;

            }

            else if (vId >= 200 and vId < 300)

            {

                cout << "Ramzan Hyperpod is being used." << endl;

            }

            else if (vId >= 300 and vId < 400)

            {

                cout << "Ramzan Time Ship is being used." << endl;

            }

            else

            {

                cout << "no vehicle..";

            }

            v->DeliveryRoute();

        }

*// v->CalTime(v->CalcDistance(0, 0, p[index]->Destx, p[index]->Desty), p[index]->weight);*

    }

    vehicle **\***AssignVehicle(Package **\***p)

    {

        vehicle \*selectedVehicle = nullptr;

*// Creating objects inside the function*

        RamzanDrone drone(101);

        RamzanHyperPod hyperPod(201);

        RamzanTimeShip timeShip(301);

**bool** droneQualifies = (drone.speed >= 10);

**bool** hyperPodQualifies = (hyperPod.speed >= 10);

**bool** timeShipQualifies = (timeShip.speed >= 10);

        if (droneQualifies && !hyperPodQualifies && !timeShipQualifies)

        {

            selectedVehicle = &drone;

        }

        else if (hyperPodQualifies && !droneQualifies && !timeShipQualifies)

        {

            selectedVehicle = &hyperPod;

        }

        else if (timeShipQualifies && !droneQualifies && !hyperPodQualifies)

        {

            selectedVehicle = &timeShip;

        }

        else if (droneQualifies && hyperPodQualifies)

        {

            vehicle &bestVehicle = ConflictResolution::resolveConflict(drone, hyperPod, p);

            selectedVehicle = &bestVehicle;

        }

        else if (droneQualifies && timeShipQualifies)

        {

            vehicle &bestVehicle = ConflictResolution::resolveConflict(drone, timeShip, p);

            selectedVehicle = &bestVehicle;

        }

        else if (hyperPodQualifies && timeShipQualifies)

        {

            vehicle &bestVehicle = ConflictResolution::resolveConflict(hyperPod, timeShip, p);

            selectedVehicle = &bestVehicle;

        }

        else if (droneQualifies && hyperPodQualifies && timeShipQualifies)

        {

            vehicle &bestVehicle = ConflictResolution::resolveConflict(drone, hyperPod, p);

            vehicle &finalChoice = ConflictResolution::resolveConflict(bestVehicle, timeShip, p);

            selectedVehicle = &finalChoice;

        }

        return selectedVehicle; *// Return the selected vehicle pointer*

    }

};

**int** main()

{

    AssignmentInfo a;

    a.WhoDidAssignment();

    Package \*\*p;

    p = new Package \*[3];

    p[0] = new Package(2, "high", 10, 10);

    p[1] = new Package(5, "medium", 15, 15);

    p[2] = new Package(3, "low", 20, 20);

    System s(p, 3);

    s.AssignVehicle(p[2]);

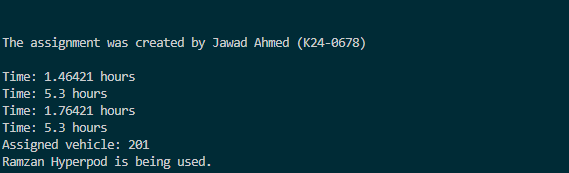
    s.command("Deliver", 2);

    delete[] p; *// Deleting the dynamically allocated memory*

    return 0;

}

**Output:**

****

**Question 4:**

**Source Code:**

#include <iostream>

using **namespace** std;

**class** AssignmentInfo

{

    string name;

    string id;

**public:**

    AssignmentInfo()

    {

        name = "Jawad Ahmed";

        id = "K24-0678";

    }

**void** WhoDidAssignment()

    {

        cout << "\n\nThe assignment was created by " << name << " (" << id << ")\n"

             << endl;

    }

};

string generateHash(string pass)

{

**long** hash = 5381;

**int** passLength = pass.length();

    for (**int** i = 0; i < passLength; i++)

    {

        hash = hash \* 33 + pass[i];

    }

    return to\_string(hash);

}

**class** User

{

**protected:**

**int** id;

    string name;

    string email;

    string hashed\_password;

    string \*PermissionList;

**int** permissions;

**public:**

    User(**int** id, string name, string email, string hashed\_password, string **\***permissionList, **int** permissions)

    {

        this->id = id;

        this->name = name;

        this->email = email;

        this->hashed\_password = hashed\_password;

        this->permissions = permissions;

        this->PermissionList = permissionList;

    }

**virtual** **void** display()

    {

        cout << "ID: " << id << endl;

        cout << "Name: " << name << endl;

        cout << "Email: " << email << endl;

        cout << "Hashed Password: " << hashed\_password << endl;

        cout << "Permission List: ";

        for (**int** i = 0; i < 5; i++)

        {

            cout << PermissionList[i] << " ";

        }

        cout << endl;

    }

**void** hasPermission(string action)

    {

        for (**int** i = 0; i < permissions; i++)

        {

            if (PermissionList[i] == action)

            {

                cout << "User has permission to perform " << action << endl;

                return;

            }

        }

    }

};

**class** Student : **public** User

{

**protected:**

    string \*assignment;

**int** assignments;

**public:**

    Student(**int** id, string name, string email, string pass, string **\***permList, **int** nPerm, string **\***assignment, **int** nAssignment) : User(id, name, email, pass, permList, nPerm)

    {

        this->assignments = nAssignment;

        this->assignment = new string[nAssignment];

        for (**int** i = 0; i < nAssignment; i++)

        {

            this->assignment[i] = assignment[i];

        }

    }

**void** submitAssignment(**int** i)

    {

        assignment[i] = "Submitted";

        cout << "Assignment " << i + 1 << " submitted successfully" << endl;

    }

**void** display() **override**

    {

        User::display();

        cout << "Assignments: ";

        for (**int** i = 0; i < assignments; i++)

        {

            cout << assignment[i] << " ";

        }

        cout << endl;

    }

};

**class** TA : **public** Student

{

    string project[2];

    Student \*\*students;

**int** nStudents;

**public:**

    TA(**int** id, string name, string email, string pass, string **\***permList, **int** nPerm, string **\***assignment, **int** nAssignment) : Student(id, name, email, pass, permList, nPerm, assignment, nAssignment)

    {

        this->nStudents = 0;

        this->students = new Student \*[10];

*// delete[] PermissionList;*

    }

**void** setProject(string ProjectName, **int** i)

    {

        if (i < 2)

            project[i] = ProjectName;

    }

    string getProject()

    {

        return project[0] + " and " + project[1];

    }

**void** addStudent(Student **\***s)

    {

        if (nStudents < 10)

        {

            students[nStudents] = s;

            nStudents++;

        }

        else

        {

            cout << "Cannot add more students" << endl;

        }

    }

**void** veiwStudents()

    {

        cout << "Students: " << endl;

        for (**int** i = 0; i < nStudents; i++)

        {

            students[i]->display();

            cout << endl;

        }

    }

};

**class** Teacher : **public** User

{

**protected:**

    string \*projects;

**int** assignedProjects;

**public:**

    Teacher(**int** id, string name, string email, string pass, string **\***permList, **int** nPerm) : User(id, name, email, pass, permList, nPerm)

    {

        projects = new string[2];

        this->assignedProjects = 0;

    }

**void** display() **override**

    {

        User::display();

        cout << "Role: Professor" << endl;

    }

**void** assignProject(TA **\***t, string ProjectName)

    {

        t->setProject(ProjectName, 1);

        cout << "Project Assigned Successfully" << endl;

    }

**void** viewAllProject(TA **\*\***t, **int** nTA)

    {

        for (**int** i = 0; i < nTA; i++)

        {

*// for(int j=0;j<2;j++){*

            cout << "Project Name: " << t[i]->getProject() << endl;

*// }*

        }

    }

};

**int** main()

{

    AssignmentInfo info;

    info.WhoDidAssignment();

*// Permissions*

    string pStudent[1] = {"submit"};

    string pTA[2] = {"View Assignments", "Manage Students"};

    string pTeacher[3] = {"View Assignments", "Manage Students", "Create Projects"};

*// Projects*

    string projects1[1] = {"Project 2"};

    string projects2[2] = {"Project 3", "Project 4"};

*// Creating Student Objects*

    Student \*s1 = new Student(3, "Alice Johnson", "alicejohnson@example.com", generateHash("qwerty456"), pStudent, 1, projects1, 1);

    Student \*s2 = new Student(4, "Bob Brown", "bobbrown@example.com", generateHash("asdfghjkl"), pStudent, 1, projects1, 1);

*// Creating TA Object*

    TA \*t1 = new TA(5, "Tom Smith", "tomsmith@example.com", generateHash("123456789"), pTA, 2, projects2, 2);

*// Creating Teacher Object*

    Teacher \*t2 = new Teacher(6, "Alice Johnson", "alicejohnson@example.com", generateHash("qwerty456"), pTeacher, 3);

*// Assign Project to TA*

    t2->assignProject(t1, "Project 3");

*// View All Projects assigned to TA*

    t2->viewAllProject(&t1, 1);

*// Submitting Assignments*

    s1->submitAssignment(0);

    s2->submitAssignment(0); *// Fixed to 0 to avoid out-of-bound error.*

*// Displaying Information*

    cout << "\nDisplaying Student Information:\n";

    s1->display();

    cout << "\nDisplaying Student Information:\n";

    s2->display();

*// Displaying TA Information*

    cout << "\nDisplaying TA Information:\n";

    t1->display();

*// Adding a student to TA's list*

    t1->addStudent(s1);

    t1->addStudent(s2);

*// Viewing Students under TA*

    t1->veiwStudents();

    return 0;

}

**Output:**

****