ASSIGNMENT 1

***QUESTION 1***

**CODE**

#include <iostream>

using namespace std;

class Pet

{

public:

    string healthStatus;

    int hungerLevel;

    int happinessLevel;

    string specialSkills[10]; // not more than 10 skills are possible

    bool adopted;

public:

    Pet() : adopted(0){};

    void displayPetDetails()

    {

        cout << "Pet Details: \n";

        cout << "Health Status: " << healthStatus << endl;

        cout << "Hunger Level: " << hungerLevel << "\n";

        cout << "happoness Level: " << happinessLevel << "\n";

        cout << "Special Skills: \n";

        int i = 0;

        while (specialSkills[i] != "")

        {

            cout << specialSkills[i] << ", ";

            ++i;

        }

    }

    void updateHappiness()

    {

        cout << "\nEnter new happiness level: ";

        cin >> happinessLevel;

        if (happinessLevel > 10)

        {

            happinessLevel = 10;

        }

    }

    void updateHealth()

    {

        cout << "\nEnter new Health Status: ";

        cin.ignore();

        cin >> healthStatus;

    }

    void updateHunger()

    {

        int a;

        cout << "\nEnter new hunger level:";

        cin >> a;

        if (a > hungerLevel)

            --happinessLevel;

        else if (a < hungerLevel)

            if (happinessLevel < 10)

                ++happinessLevel;

        hungerLevel = a;

    }

};

class Adopter

{

public:

    string adopterName;

    string adopterMobileNum;

    struct record

    {

        string \*healthStatus;

        int \*hungerLevel;

        int \*happinessLevel;

        string specialSkills[100];

        bool adopted;

    };

    record adoptedPetRecords[10]; // can't adopt more than 10

    int count;

public:

    Adopter(string name, string num) : adopterName(name), adopterMobileNum(num), count(0) {}

    void adoptPet(Pet &newpet)

    {

        if (newpet.adopted == 1)

            cout << "\nSorry the pet has been adopted ";

        else

        {

            newpet.adopted = 1;

            adoptedPetRecords[count].happinessLevel = &newpet.happinessLevel;

            adoptedPetRecords[count].hungerLevel = &newpet.hungerLevel;

            adoptedPetRecords[count].healthStatus = &newpet.healthStatus;

            for (int i = 0; i < 10; i++)

            {

                adoptedPetRecords[count].specialSkills[i] = newpet.specialSkills[i];

            }

            adoptedPetRecords[count].adopted = 1;

            count++;

        }

    }

    void returnPet(Pet &returnPet)

    {

        for (int i = 0; i < count; i++)

        {

            if (\*(adoptedPetRecords[i].happinessLevel) == returnPet.happinessLevel)

            {

                returnPet.adopted = 0;

                adoptedPetRecords[i].adopted = 0;

                adoptedPetRecords[i].happinessLevel = nullptr;

                adoptedPetRecords[i].healthStatus = nullptr;

                adoptedPetRecords[i].hungerLevel = nullptr;

                for (int j = 0; j < 10; j++)

                {

                    adoptedPetRecords[i].specialSkills[j] = "";

                }

                count--;

                cout << "\n The pet has been returned";

            }

            else if (returnPet.adopted == 1)

                cout << "\n This pet record not found, it was not adopted by you";

        }

    }

    void displayAdoptedPets()

    {

        for (int i = 0; i < count; i++)

        {

            cout << "\n\nPet " << i + 1 << " Details: \n";

            cout << "Health Status: " << \*(adoptedPetRecords[i].healthStatus) << endl;

            cout << "Hunger Level: " << \*(adoptedPetRecords[i].hungerLevel) << "\n";

            cout << "happoness Level: " << \*(adoptedPetRecords[i].happinessLevel) << "\n";

            cout << "Special Skills: \n";

            for (int j = 0; j < 10; j++)

            {

                cout << adoptedPetRecords[i].specialSkills[j] << ", ";

            }

        }

    }

};

int main()

{

    cout << "23k0800 Muhammad Mufeez" << endl;

    string name, phone;

    int choice;

    cout << "Enter your name: " << endl;

    cin >> name;

    cout << "Enter your phone: ";

    cin >> phone;

    Adopter mufeez(name, phone);

    Pet cat, dog, horse;

    mufeez.adoptPet(cat);

    cout << "Play with your pet!";

    cat.updateHappiness();

    cat.updateHealth();

    cat.updateHunger();

    cat.displayPetDetails();

    cat.updateHappiness();

    cat.updateHealth();

    cat.updateHunger();

    cat.displayPetDetails();

    mufeez.adoptPet(dog);

    cout << "Play with your pet!";

    dog.updateHappiness();

    dog.updateHealth();

    dog.updateHunger();

    dog.displayPetDetails();

    dog.updateHappiness();

    dog.updateHealth();

    dog.updateHunger();

    dog.displayPetDetails();

    mufeez.displayAdoptedPets();

    mufeez.returnPet(cat);

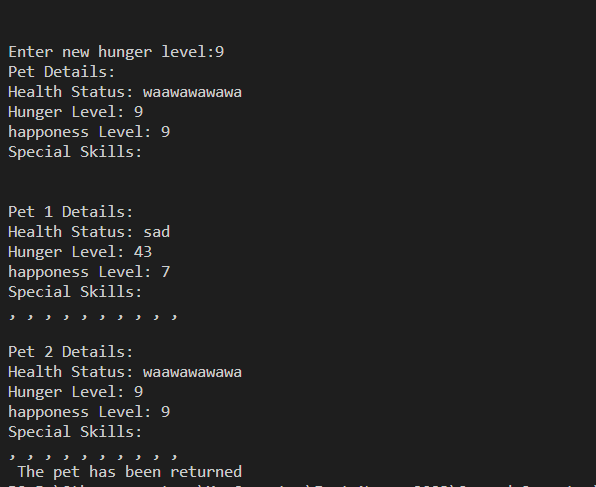
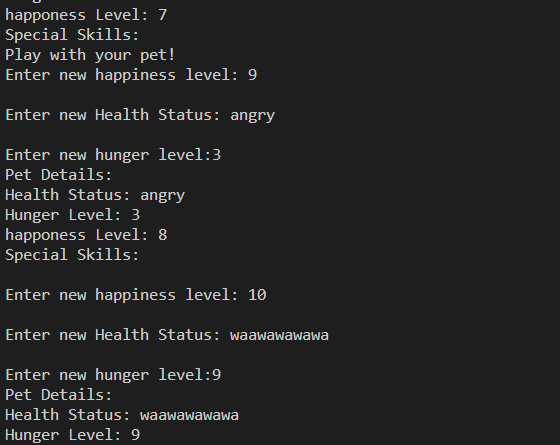
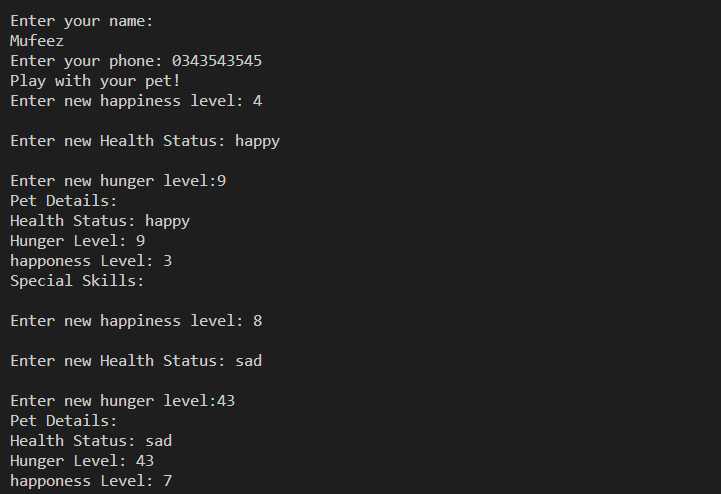
    mufeez.returnPet(dog);

    mufeez.displayAdoptedPets();

    return 0;

}

**OUTPUT**



***QUESTION 2***

**CODE**

#include <iostream>

#include <cmath>

using namespace std;

class table

{

private:

    int seats;

    int occupiedSeats;

    int freeSeats;

    bool clean;

public:

    int GetSeats() const

    {

        return seats;

    }

    int GetOccupiedSeats() const

    {

        return occupiedSeats;

    }

    void SetOccupiedSeats(int occupiedSeats)

    {

        this->occupiedSeats = occupiedSeats;

        freeSeats = seats - occupiedSeats;

    }

    int GetFreeSeats() const

    {

        return freeSeats;

    }

    void SetFreeSeats(int freeSeats)

    {

        this->freeSeats = freeSeats;

        occupiedSeats = seats - freeSeats;

    }

    bool GetClean() const

    {

        return clean;

    }

    void SetClean(bool clean)

    {

        this->clean = clean;

    }

    table() : seats(4), occupiedSeats(0), freeSeats(4), clean(1) {}

    table(int capacity) : occupiedSeats(0), clean(1)

    {

        if (capacity == 4 || capacity == 8)

        {

            seats = capacity;

        }

        else

        {

            seats = round(capacity / 4.0) \* 4.0;

            // if user enters number more than 12 than the seats will be 12, thats why the following will handle this error

            if (seats > 8)

                seats = 8;

        }

        freeSeats = seats;

    }

    void usingTheTable()

    {

        cout << "\nStudents are using the table";

    }

    void havingLunchOnTable()

    {

        cout << "\nStudents are having lunch on table";

        this->SetClean(0);

    }

    void leavingTheTable()

    {

        cout << "\nStudents are about to leave";

    }

    void cleaningTheTable()

    {

        cout << "\nTable is being  cleaned";

        this->SetClean(1);

    }

};

void OccupyTable(table \*tables, int size)

{

    bool found = false;

    for (int i = 0; i < 5; ++i)

    {

        if (tables[i].GetOccupiedSeats() == 0 && tables[i].GetSeats() >= size)

        {

            cout << "\nTable found, you have been assigned the table no " << i + 1 << ", please be seated, seating capacity of table is: " << tables[i].GetSeats();

            tables[i].SetOccupiedSeats(size);

            found = true;

            break;

        }

    }

    if (found == false)

    {

        cout << "\n Sorry none of the tables is free" << endl;

    }

}

void emptyTable(int number, table \*tables)

{

    // tables[number].SetClean(1);

    tables[number].SetOccupiedSeats(0);

    cout << "\n Table is empty now";

}

int main()

{

    cout << "23k0800 Muhammad Mufeez" << endl;

    table tables[5];

    for (int i = 0; i < 5; i++)

    {

        if (i < 2)

            tables[i] = table(7);

        else

            tables[i] = table(2);

    }

    OccupyTable(tables, 4);

    OccupyTable(tables, 6);

    tables[0].usingTheTable();

    tables[0].havingLunchOnTable();

    tables[0].leavingTheTable();

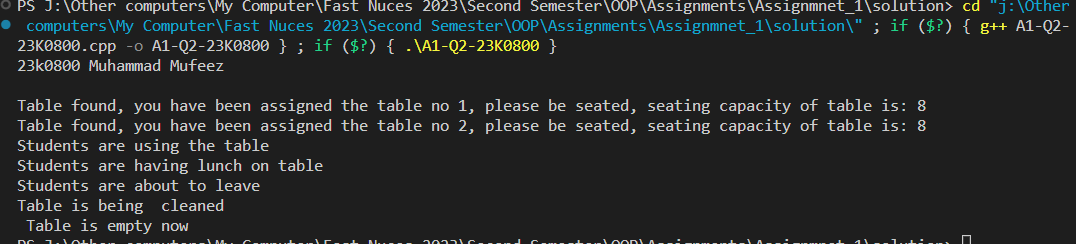
    tables[0].cleaningTheTable();

    emptyTable(1, tables);

    return 0;

}

**OUTPUT**



***QUESTION 3***

**CODE**

#include <iostream>

#include <string>

using namespace std;

class ChessPiece

{

private:

    string name;

    string color;

    string symbol;

public:

    string GetName() const

    {

        return name;

    }

    void SetName(string name)

    {

        this->name = name;

    }

    string GetColor() const

    {

        return color;

    }

    void SetColor(string color)

    {

        this->color = color;

    }

    string GetSymbol() const

    {

        return symbol;

    }

    void SetSymbol(string symbol)

    {

        this->symbol = symbol;

    }

    ChessPiece()

    {

        name = "pawn";

        color = "white";

        symbol = "p";

    }

    ChessPiece(string name, string color, string symbol)

    {

        this->name = name;

        this->color = color;

        this->symbol = symbol;

    }

};

class ChessBoard

{

public:

    ChessPiece \*table[8][8];

public:

    ChessBoard()

    {

        string names[] = {"rook", "knight", "bishop", "queen", "king", "bishop", "knight", "rook"};

        string symbols\_white[] = {"r", "n", "b", "q", "k", "b", "n", "r"};

        string symbols\_black[] = {"R", "N", "B", "Q", "K", "B", "N", "R"};

        for (int i = 0; i < 8; ++i)

        {

            for (int j = 0; j < 8; ++j)

            {

                if (i == 0)

                {

                    table[i][j] = new ChessPiece(names[j], "black", symbols\_black[j]);

                }

                else if (i == 1)

                {

                    table[i][j] = new ChessPiece("pawn", "black", "P");

                }

                else if (i == 6)

                {

                    table[i][j] = new ChessPiece();

                }

                else if (i == 7)

                {

                    table[i][j] = new ChessPiece(names[j], "white", symbols\_white[j]);

                }

                else

                {

                    table[i][j] = nullptr;

                }

            }

        }

    }

    void display()

    {

        for (int i = 0; i < 8; ++i)

        {

            for (int j = 0; j < 8; ++j)

            {

                if (table[i][j] == nullptr)

                    cout << "-\t";

                else

                    cout << table[i][j]->GetSymbol() << "\t";

            }

            cout << "\n";

        }

    }

    bool movePiece(string s, string d)

    {

        int y1 = s[0] - 'a';

        int x1 = s[1] - '1';

        x1 = 7 - x1;

        int y2 = d[0] - 'a';

        int x2 = d[1] - '1';

        x2 = 7 - x2;

        if (table[x1][y1]->GetName() == "knight")

        {

            if (((x2 == x1 + 2) && ((y2 == y1 + 1) || (y2 == y1 - 1))) ||

                ((x2 == x1 - 2) && ((y2 == y1 + 1) || (y2 == y1 - 1))) ||

                ((y2 == y1 + 2) && ((x2 == x1 + 1) || (x2 == x1 - 1))) ||

                ((y2 == y1 - 2) && ((x2 == x1 + 1) || (x2 == x1 - 1))))

            {

                if (table[x2][y2] != nullptr)

                    return false;

                else

                    return true;

            }

            else

                return false;

        }

        else if (table[x1][y1]->GetName() == "pawn")

        {

            if (x1 == x2 && ((y2 == y1 + 1) || (y2 == y1 + 2) || (y2 == y1 - 1) || (y2 == y1 - 2)))

            {

                if (table[x2][y2] != nullptr)

                    return false;

                else

                    return true;

            }

            else

                return false;

        }

        else if (table[x1][y1]->GetName() != "knight" && table[x1][y1]->GetName() != "pawn")

        {

            cout << "\n\tFirst move can be done only on pawn or knight, so write initial position of any of them" << endl;

            return false;

        }

        else

            return false;

    }

    ~ChessBoard()

    {

        for (int i = 0; i < 8; i++)

        {

            for (int j = 0; j < 8; ++j)

            {

                delete table[i][j];

                table[i][j] = nullptr;

            }

        }

    }

};

int main()

{

    ChessBoard c1;

    c1.display();

    bool hello;

    hello = c1.movePiece("b8", "a6");

    if (hello == true)

        cout << "\n successful" << endl;

    else

        cout << "\n failed" << endl;

    hello = c1.movePiece("b8", "d7");

    if (hello == true)

        cout << "\n successful" << endl;

    else

        cout << "\n failed" << endl;

    cout << "\n\n\n";

    string start, destination;

    cout << "Enter staarting position: " << endl;

    cin >> start;

    cout << "Enter destination: " << endl;

    cin >> destination;

    hello = c1.movePiece(start, destination);

    if (hello == true)

        cout << "\n successful" << endl;

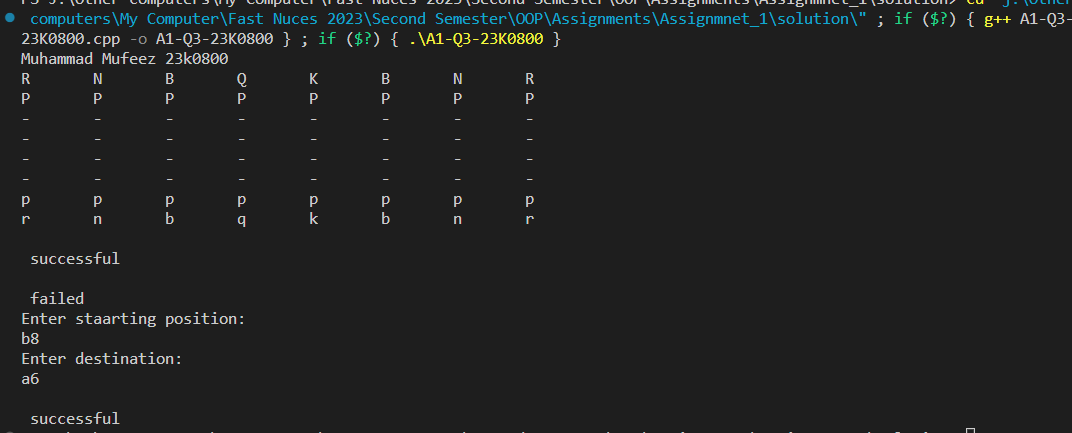
    else

        cout << "\n failed" << endl;

    return 0;

}

**OUTPUT**



***QUESTION 4***

**CODE**

#include <iostream>

#include <cmath>

using namespace std;

class RollerCoaster

{

private:

    string Name;

    int height;

    int length;

    int speed;

    int capacity;

    int currentNumRiders;

    bool rideInProgress;

    int id, sp; // it was not required, but in the functions of acceleration and deaceleration, i had to use these

public:

    RollerCoaster() : Name("roller coaster"), height(500), length(2000), capacity(20), rideInProgress(false), speed(0) {}

    RollerCoaster(string Name, int height, int length, int capacity) : speed(0)

    {

        this->Name = Name;

        this->height = height;

        this->length = length;

        if (capacity < 3)

        {

            cout << "Enter capacity greater than 3: ";

            cin >> capacity;

        }

        if (capacity % 2 == 0 || capacity % 3 == 0)

            this->capacity = capacity;

        else

            this->capacity = ++capacity;

        rideInProgress = false;

    }

    int loadRiders()

    {

        int number;

        cout << "Ener number of riders: ";

        cin >> number;

        if (!rideInProgress)

        {

            cout << "\nRide is not in progress, entering riders";

            if (number > capacity)

            {

                currentNumRiders = capacity;

                return (number - capacity);

            }

            else

            {

                currentNumRiders = number;

                return 0;

            }

        }

        else

            cout << "\nRide is in progress";

    }

    int startRide()

    {

        if (rideInProgress)

        {

            cout << "\n Ride can not be started as it is in prہgress";

            return -1;

        }

        else if ((capacity - currentNumRiders) == 0)

        {

            cout << "\nRide has been started";

            rideInProgress = true;

            return 0;

        }

        else

        {

            cout << "\nAll seats are not filled";

            return (capacity - currentNumRiders);

        }

    }

    void stopRide()

    {

        if (rideInProgress)

        {

            cout << "\nRide was in progress, but it's been stopped now";

            rideInProgress = false;

        }

        else

            cout << "\nRide is already stopped";

    }

    void unloadRiders()

    {

        if (currentNumRiders == 0)

        {

            cout << "\nNo riders to unload.";

        }

        else if (!rideInProgress)

        {

            currentNumRiders = 0;

            cout << "\nPassengers unloaded";

        }

        else

            cout << "\n cant unload passengers, as ride is in progress";

    };

    void accelerateRollerCoster()

    {

        if (rideInProgress)

        {

            cout << "Enter your id: ";

            cin >> id;

            sp = returnLastNonZero(id);

            cout << "\n Speed has been increase by :" << sp;

            speed += sp;

            cout << "\n Current speed is: " << speed;

        }

        else

            cout << "Ride is not started yet";

    };

    void decelerateRollerCoaster()

    {

        if (rideInProgress)

        {

            cout << "\n Enter your id: ";

            cin >> id;

            sp = returnFirstNonZero(id);

            speed -= sp;

            cout << "\n Speed has been decreased by :" << sp;

            cout << "\n Current speed is: " << speed;

        }

        else

            cout << "Ride is not started yet";

    };

    int returnLastNonZero(int id)

    {

        if (id % 10 != 0)

        {

            return (id % 10);

        }

        else if (id == 0)

        {

            cout << "\n entered id is not correct";

            return 0;

        }

        else

        {

            returnLastNonZero(id / 10);

        }

    }

    int returnFirstNonZero(int id, int n = 1000)

    {

        if (id == 0)

        {

            cout << "\nIncorrect id entered";

            return 0;

        }

        else if (id / n != 0)

            return (id / n);

        else

            return returnFirstNonZero(id, (n / 10));

    }

    string GetName() const

    {

        return Name;

    }

    void SetName(string Name)

    {

        this->Name = Name;

    }

    int GetHeight() const

    {

        return height;

    }

    void SetHeight(int height)

    {

        this->height = height;

    }

    int GetLength() const

    {

        return length;

    }

    void SetLength(int length)

    {

        this->length = length;

    }

    int GetSpeed() const

    {

        return speed;

    }

    void SetSpeed(int speed)

    {

        this->speed = speed;

    }

    int GetCapacity() const

    {

        return capacity;

    }

    void SetCapacity(int capacity)

    {

        if (capacity < 3)

        {

            cout << "Enter capacity greater than 3: ";

            cin >> capacity;

        }

        if (capacity % 2 == 0 || capacity % 3 == 0)

            this->capacity = capacity;

        else

            this->capacity = ++capacity;

    }

    int GetCurrentNumRiders() const

    {

        return currentNumRiders;

    }

    void SetCurrentNumRiders(int currentNumRiders)

    {

        this->currentNumRiders = currentNumRiders;

    }

    bool GetRideInProgress() const

    {

        return rideInProgress;

    }

    void SetRideInProgress(bool rideInProgress)

    {

        this->rideInProgress = rideInProgress;

    }

};

int main()

{

    cout << "23k0800 Muhammad Mufeez" << endl;

    RollerCoaster R1, R2("Tornado", 800, 3500, 49);

    int choice = 0;

    while (choice != 99)

    {

        cout << "\n\nRollerCoaster Menu:\n";

        cout << "1. Load riders onto the roller coaster.\n";

        cout << "2. Start the roller coaster ride.\n";

        cout << "3. Stop the roller coaster ride.\n";

        cout << "4. Unload riders from the roller coaster.\n";

        cout << "5. Accelerate the roller coaster.\n";

        cout << "6. Decelerate the roller coaster.\n";

        cout << "99. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            int n = R2.loadRiders();

            cout << "\nremaining riders are: " << n << endl;

            break;

        }

        case 2:

        {

            int n = R2.startRide();

            if (n > 0)

                cout << "\nFill in the remaining seats: " << n << endl;

            break;

        }

        case 3:

            R2.stopRide();

            break;

        case 4:

            R2.unloadRiders();

            break;

        case 5:

            R2.accelerateRollerCoster();

            break;

        case 6:

            R2.decelerateRollerCoaster();

            break;

        case 99:

            cout << "Exiting...\n";

            break;

        default:

            cout << "Invalid choice. Please try again.\n";

            break;

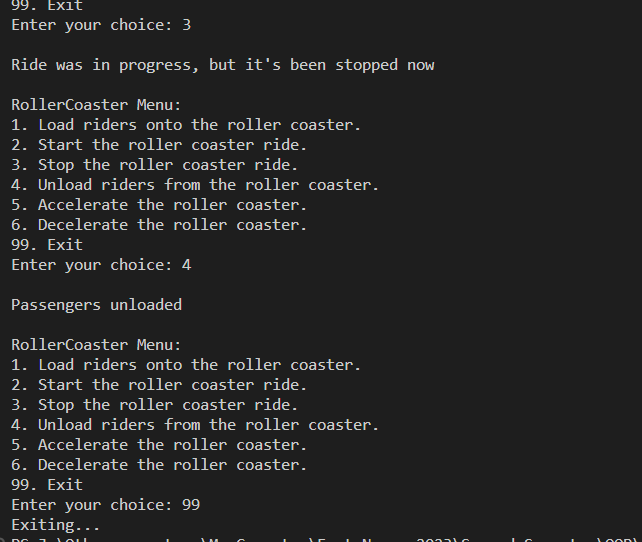
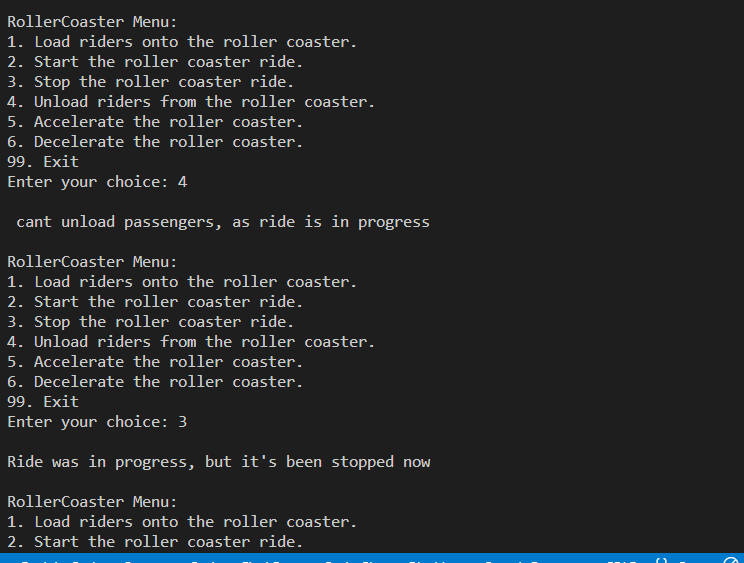
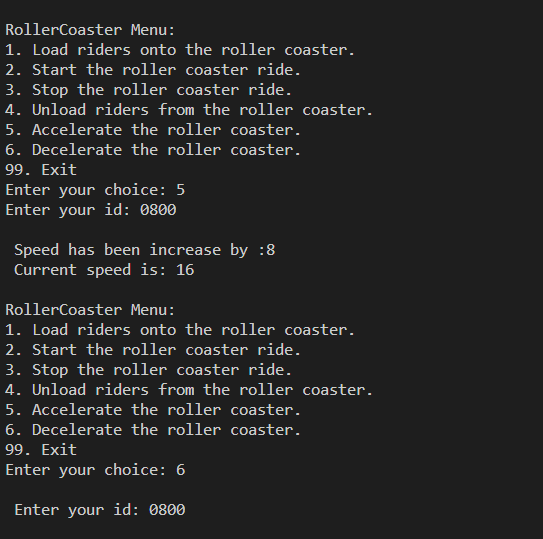
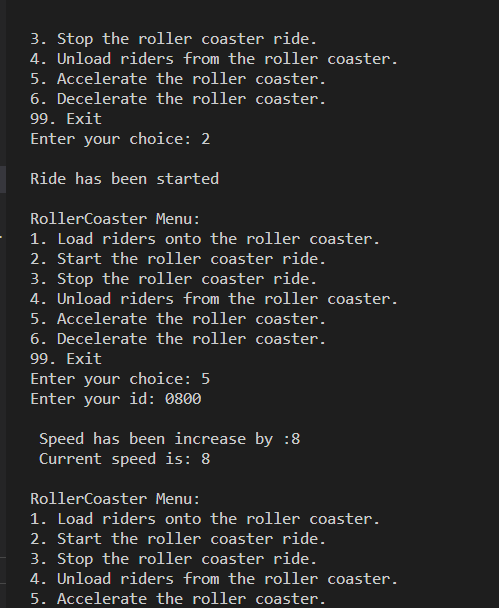
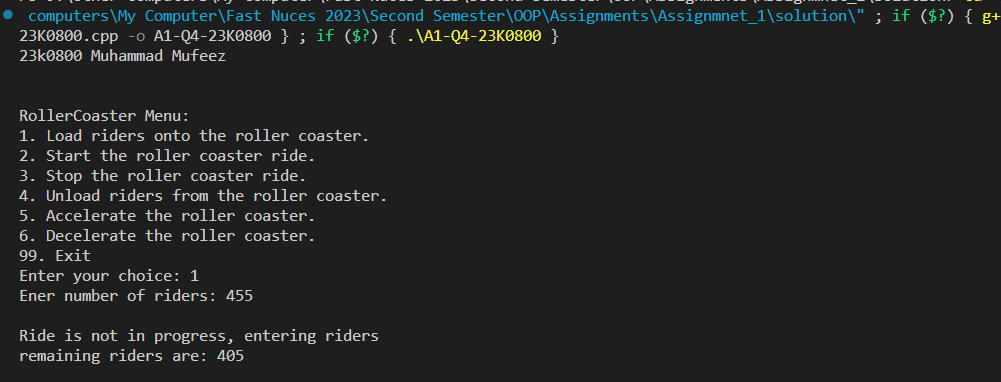
        }

    }

    return 0;

}

**OUTPUT**



***QUESTION 5***

**CODE**

#include <iostream>

#include <string>

using namespace std;

class BOGOCoupon

{

private:

    string coupon\_code;

    int valid\_from;

    int valid\_until;

    string restuarant\_code;

public:

    BOGOCoupon(){};

    string GetCoupon\_code() const

    {

        return coupon\_code;

    }

    void SetCoupon\_code(string coupon\_code)

    {

        this->coupon\_code = coupon\_code;

    }

    int GetValid\_from() const

    {

        return valid\_from;

    }

    void SetValid\_from(int valid\_from)

    {

        this->valid\_from = valid\_from;

    }

    int GetValid\_until() const

    {

        return valid\_until;

    }

    void SetValid\_until(int valid\_until)

    {

        this->valid\_until = valid\_until;

    }

    string GetRestuarant\_code() const

    {

        return restuarant\_code;

    }

    void SetRestuarant\_code(string restuarant\_code)

    {

        this->restuarant\_code = restuarant\_code;

    }

    BOGOCoupon(string c, int vf, int vu, string rc) : valid\_from(vf), valid\_until(vu), restuarant\_code(rc), coupon\_code(c) {}

    bool is\_valid(int current\_date, string restuarantCode)

    {

        return ((current\_date >= valid\_from) && (current\_date <= valid\_until)) && (restuarant\_code == restuarantCode);

    }

};

class User

{

private:

    string name;

    int age;

    string mobile\_number;

public:

    BOGOCoupon coupons\_list[50]; // user can not have more then 50 coupons

    BOGOCoupon redeemed\_coupons\_list[50];

    int coupons;

    int redeemedCoupons;

public:

    string GetName()

    {

        return name;

    }

    void SetName(string name)

    {

        this->name = name;

    }

    int GetAge() const

    {

        return age;

    }

    void SetAge(int age)

    {

        this->age = age;

    }

    string GetMobile\_number() const

    {

        return mobile\_number;

    }

    void SetMobile\_number(string mobile\_number)

    {

        this->mobile\_number = mobile\_number;

    }

    User(string Name, int Age, string MobileNumber) : name(Name), age(Age), mobile\_number(MobileNumber), coupons(0), redeemedCoupons(0) {}

    void accumulate\_coupon(string code, int valid\_from, int valid\_until, string restuarant\_code)

    {

        coupons\_list[coupons].SetCoupon\_code(code);

        coupons\_list[coupons].SetValid\_from(valid\_from);

        coupons\_list[coupons].SetValid\_until(valid\_until);

        coupons\_list[coupons].SetRestuarant\_code(restuarant\_code);

        cout << "\n\tCoupon accumulated successfully" << endl;

        ++coupons;

    }

    void has\_valid\_coupon(string restuarantCode)

    {

        int count = 0;

        for (int i = 0; i < coupons; ++i)

        {

            if (coupons\_list[i].GetRestuarant\_code() == restuarantCode)

            {

                if (coupons\_list[i].is\_valid(20240229, restuarantCode))

                {

                    cout << "\n Valid Coupon with code: " << coupons\_list[i].GetCoupon\_code();

                    count++;

                }

            }

        }

        if (count == 0)

            cout << "\n There is no valid coupon " << endl;

    };

    bool redeem\_coupon(string coupon\_code, int current\_date, string restuarantCode)

    {

        for (int i = 0; i < redeemedCoupons; ++i)

        {

            if (redeemed\_coupons\_list[i].GetCoupon\_code() == coupon\_code)

            {

                cout << "\n Coupon has been redeemed before" << endl;

                return false;

            }

        }

        for (int i = 0; i < coupons; i++)

        {

            if (coupons\_list[i].GetCoupon\_code() == coupon\_code)

            {

                if (!coupons\_list[i].is\_valid(current\_date, restuarantCode))

                {

                    cout << "\n Invalid Coupon " << endl;

                    return false;

                }

                else

                {

                    cout << this->GetName() << "has  Valid Coupon" << endl;

                    redeemed\_coupons\_list[redeemedCoupons].SetCoupon\_code(coupons\_list[i].GetCoupon\_code());

                    redeemed\_coupons\_list[redeemedCoupons].SetRestuarant\_code(coupons\_list[i].GetRestuarant\_code());

                    redeemed\_coupons\_list[redeemedCoupons].SetValid\_from(coupons\_list[i].GetValid\_from());

                    redeemed\_coupons\_list[redeemedCoupons].SetValid\_until(coupons\_list[i].GetValid\_until());

                    ++redeemedCoupons;

                    return true;

                }

            }

        }

        cout << "\n Enter the correct coupon code associated with the restuarant code" << endl;

        return false;

    };

};

class Restuarant

{

public:

    int static coupons\_redeemed\_count;

    struct orderList

    {

        int quantity;

        int price;

        string name;

    };

    string coupon;

    int count;

    int total;

    int menu;

    string prefix\_restuarant;

private:

    string restuarant\_name;

    string location;

public:

    string menu\_list[3]; // in the given situation, there are only 3 items in both restuarants

    int price\_list[3];   // in pkr, thats why int

    string valid\_coupon\_code\_list[50];

public:

    Restuarant(string name, string located, string prefix) : restuarant\_name(name), location(located), count(0), total(0), prefix\_restuarant(prefix), menu(0) {}

    string GetRestuarant\_name() const

    {

        return restuarant\_name;

    }

    void SetRestuarant\_name(string restuarant\_name)

    {

        this->restuarant\_name = restuarant\_name;

    }

    string GetLocation() const

    {

        return location;

    }

    void SetLocation(string location)

    {

        this->location = location;

    }

    void generate\_bill(User &customer)

    {

        this->display\_menu();

        cout << "\nHow man distinct items you wanna order? (not more than 3)" << endl;

        cin >> count;

        orderList listOfBooking[count];

        int choice;

        for (int i = 0; i < count; ++i)

        {

            cout << "\n For item no :" << i + 1 << endl;

            cout << "Enter  item number :" << endl;

            cin >> choice;

            listOfBooking[i].name = menu\_list[choice - 1];

            cout << "Enter  quantity :" << endl;

            cin >> listOfBooking[i].quantity;

            listOfBooking[i].price = price\_list[choice - 1] \* listOfBooking[i].quantity;

            total += listOfBooking[i].price;

        }

        cout << "\n Items Ordered are given below: " << endl;

        cout << "Name"

             << "\t"

             << "Quantity"

             << "\t"

             << "Price" << endl;

        for (int i = 0; i < count; ++i)

        {

            cout << listOfBooking[i].name << "\t" << listOfBooking[i].quantity << "\t" << listOfBooking[i].price << endl;

        }

        cout << "Total : " << total << endl;

        cout << "Do you have any discount code? if yes than enter the code : " << endl;

        cin >> coupon;

        bool applicable = customer.redeem\_coupon(coupon, 20240229, prefix\_restuarant);

        if (applicable)

        {

            coupons\_redeemed\_count++;

            cout << "\n Congratulations, coupon is valid and BOGO offer applied on the first item" << endl;

            listOfBooking[0].quantity += 1;

            for (int i = 0; i < count; ++i)

            {

                cout << listOfBooking[i].name << "\t" << listOfBooking[i].quantity << "\t" << listOfBooking[i].price << endl;

                cout << "Total : " << total << endl;

            }

        }

        else

        {

            cout << "\n Sorry, the coupon code is invalid" << endl;

        }

    };

    void display\_menu()

    {

        cout << "\n\nDisplaying Menu of " << restuarant\_name << ": " << endl;

        for (int i = 0; i < 3; i++)

        {

            cout << menu\_list[i] << "\tPrice: " << price\_list[i] << endl;

        }

    };

    void updateMenu(string name, int price)

    {

        menu\_list[menu] = name;

        price\_list[menu] = price;

        ++menu;

    }

};

int Restuarant::coupons\_redeemed\_count = 0;

int main()

{

    cout << "23k0800 Muhammad Mufeez" << endl;

    User mufeez("Mufeez", 20, "03230434423");

    // accumulating coupons to user

    mufeez.accumulate\_coupon("FH-BOGO-12345", 20230229, 20240229, "FH");

    mufeez.accumulate\_coupon("FH-BOGO-12345", 20230229, 20220229, "FH");

    mufeez.accumulate\_coupon("PB-BOGO-12345", 20230229, 20240229, "PB");

    mufeez.accumulate\_coupon("SH-BOGO-12345", 20230229, 20240229, "SH");

    mufeez.accumulate\_coupon("OH-BOGO-12345", 20230229, 20240229, "OH");

    // testing has valid  coupon method

    mufeez.has\_valid\_coupon("SH");

    // creating restuarant

    Restuarant FoodHaven("Food Haven", "City Center", "FH");

    Restuarant PixelBites("Pixel Bites", "Cyber Street", "PB");

    // adding items to the food haven's menu

    FoodHaven.updateMenu("Sushi", 100);

    FoodHaven.updateMenu("Pad Thai", 200);

    FoodHaven.updateMenu("Mango Tango", 300);

    // adding items to the pixel bites menu

    PixelBites.updateMenu("Binary Burger", 1000);

    PixelBites.updateMenu("Quantum Quinoa", 2000);

    PixelBites.updateMenu("Data Donuts", 3000);

    FoodHaven.generate\_bill(mufeez);

    PixelBites.generate\_bill(mufeez);

    mufeez.has\_valid\_coupon("PB");

    return 0;

}

**OUTPUT**

