Solution for Assignment # 1:

Question # 1: Adopter and Pet one

#include <iostream>

#include <string>

using namespace std;

class Pet {

private:

string healthStatus;

int hungerLevel;

int happinessLevel;

string specialSkills;

public:

Pet(string health="Null", int hunger=0, int happiness=0, string skills="Null")

: healthStatus(health), hungerLevel(hunger), happinessLevel(happiness), specialSkills(skills) {}

void displayPetDetails() {

cout << "Pet Details:" << endl;

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

cout << "Hunger Level: " << hungerLevel << endl;

cout << "Happiness Level: " << happinessLevel << endl;

cout << "Special Skills: " << specialSkills << endl;

}

void updateHappiness() {

if (hungerLevel > 5) {

happinessLevel -= 1;

} else {

happinessLevel += 1;

if (happinessLevel > 10) {

happinessLevel = 10;

}

}

}

void updateHealth(string a) {

healthStatus=a;

}

void updateHunger(int a) {

hungerLevel=a;

updateHappiness();

}

void feedPet() {

hungerLevel -= 1;

if (hungerLevel < 0) {

hungerLevel = 0;

}

updateHappiness();

}

};

class Adopter {

private:

string adopterName;

string adopterMobileNum;

Pet adoptedPetRecords[5];

int PetCount;

public:

Adopter(string name, string mobileNum)

: adopterName(name), adopterMobileNum(mobileNum) {

PetCount=0;

}

void adoptPet(Pet pet) {

if(PetCount<5)

{ adoptedPetRecords[PetCount] = pet;

cout << adopterName << " has adopted a pet!" << endl;

PetCount+=1;;

return;

}

cout << "Sorry, " << adopterName << ", you cannot adopt more pets." << endl;

}

Pet returnPet(int petIndex) {

if (petIndex >= 0 && petIndex < 5 ) {

Pet temp;

cout << adopterName << " has returned a pet." << endl;

adoptedPetRecords[petIndex] = temp;

} else {

cout << "Invalid pet index or no pet at the specified index." << endl;

}

}

void displayAdoptedPets() {

cout << "Adopted Pets by " << adopterName << ":" << endl;

for (int i = 0; i < PetCount; ++i) {

adoptedPetRecords[i].displayPetDetails();

}

}

};

int main() {

Pet cat("Healthy", 3, 8, "Climbing, Purring");

Pet dog("Sick", 6, 6, "Fetching, Guarding");

Adopter adopter1("Sana", "123-456-7890");

Adopter adopter2("Ali", "987-654-3210");

adopter1.adoptPet(cat);

adopter2.adoptPet(dog);

cat.feedPet();

dog.feedPet();

adopter1.displayAdoptedPets();

adopter2.displayAdoptedPets();

return 0;

}

Question # 2: Dhaba One

#include <iostream>

using namespace std;

class Table {

private:

const int \_capacity;

int \_occupiedSeats;

bool \_clean;

public:

Table(int capacity = 4) : \_capacity(capacity), \_occupiedSeats(0), \_clean(true) {}

int getCapacity() const {

return \_capacity;

}

int getOccupiedSeats() const {

return \_occupiedSeats;

}

int getFreeSeats() const {

return \_capacity - \_occupiedSeats;

}

bool isClean() const {

return \_clean;

}

void useTable(int groupSize) {

if (\_clean && groupSize <= \_capacity) {

\_occupiedSeats = groupSize;

\_clean = false;

cout << "Table with capacity " << \_capacity << " assigned to a group of " << groupSize << ".\n";

}

}

void haveLunch() {

if (!\_clean) {

cout << "Having lunch on the table.\n";

\_clean = false;

}

}

void leaveTable() {

cout << "Leaving the table.\n";

\_occupiedSeats = 0;

}

void cleanTable() {

if (\_occupiedSeats == 0) {

cout << "Cleaning the table.\n";

\_clean = true;

}

else {

cout << "Cannot clean the table while people are still seated.\n";

}

}

};

void OccupyTable(Table tables[], int numTables, int groupSize) {

for (int i = 0; i < numTables; ++i) {

if (tables[i].getFreeSeats() >= groupSize) {

tables[i].useTable(groupSize);

break;

}

}

}

void EmptyTable(Table tables[], int tableIndex) {

tables[tableIndex].leaveTable();

}

int main() {

const int numTables = 5;

Table tables[numTables] = { Table(8), Table(8), Table(4), Table(4), Table(4) };

OccupyTable(tables, numTables, 4); // Assume this is table 1

OccupyTable(tables, numTables, 6); // Assume this is table 2

tables[0].useTable(4); // Using table 1

tables[0].haveLunch(); // Having lunch on table 1

tables[0].leaveTable(); // Leaving table 1

tables[0].cleanTable(); // Cleaning table 1

EmptyTable(tables, 1); // Emptying table 2

return 0;

}

Question # 3: Chess one

#include <iostream>

#include <iomanip>

using namespace std;

class ChessPiece {

private:

string name;

char symbol;

string color;

public:

// Default Constructor

ChessPiece() : name("Pawn"), symbol('p'), color("white") {}

// Parameterized Constructor

ChessPiece(string \_name, char \_symbol, string \_color)

: name(\_name), symbol(\_symbol), color(\_color) {}

// Getter functions

string getName() const {

return name;

}

char getSymbol() const {

return symbol;

}

string getColor() const {

return color;

}

// Setter functions

void setName(string \_name) {

name = \_name;

}

void setSymbol(char \_symbol) {

symbol = \_symbol;

}

void setColor(string \_color) {

color = \_color;

}

};

class ChessBoard {

private:

ChessPiece\* board[8][8];

public:

// Default Constructor

ChessBoard() {

// Initialize the board with initial game state

// White pieces

board[0][0] = new ChessPiece("Rook", 'R', "white");

board[0][1] = new ChessPiece("Knight", 'N', "white");

board[0][2] = new ChessPiece("Bishop", 'B', "white");

board[0][3] = new ChessPiece("Queen", 'Q', "white");

board[0][4] = new ChessPiece("King", 'K', "white");

board[0][5] = new ChessPiece("Bishop", 'B', "white");

board[0][6] = new ChessPiece("Knight", 'N', "white");

board[0][7] = new ChessPiece("Rook", 'R', "white");

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

board[1][i] = new ChessPiece("Pawn", 'p', "white");

// Black pieces

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

board[6][i] = new ChessPiece("Pawn", 'P', "black");

board[7][0] = new ChessPiece("Rook", 'r', "black");

board[7][1] = new ChessPiece("Knight", 'n', "black");

board[7][2] = new ChessPiece("Bishop", 'b', "black");

board[7][3] = new ChessPiece("Queen", 'q', "black");

board[7][4] = new ChessPiece("King", 'k', "black");

board[7][5] = new ChessPiece("Bishop", 'b', "black");

board[7][6] = new ChessPiece("Knight", 'n', "black");

board[7][7] = new ChessPiece("Rook", 'r', "black");

// Empty spaces

for (int i = 2; i < 6; i++) {

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

board[i][j] = nullptr;

}

}

}

// Destructor

~ChessBoard() {

for (int i = 0; i < 8; i++) {

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

delete board[i][j];

}

}

}

// Display the current state of the board

void display() {

cout << " a b c d e f g h\n";

for (int i = 0; i < 8; i++) {

cout << 8 - i << " ";

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

if (board[i][j] != nullptr) {

cout << board[i][j]->getSymbol() << " ";

}

else {

cout << ". ";

}

}

cout << 8 - i << "\n";

}

cout << " a b c d e f g h\n";

}

// Move a piece from source to destination

bool movePiece(string source, string destination) {

int sourceRow = 8 - (source[1] - '0');

int sourceCol = source[0] - 'a';

int destRow = 8 - (destination[1] - '0');

int destCol = destination[0] - 'a';

if (board[sourceRow][sourceCol] == nullptr) {

cout << "No piece at source position.\n";

return false;

}

if (board[destRow][destCol] != nullptr) {

cout << "Destination position already occupied.\n";

return false;

}

ChessPiece\* piece = board[sourceRow][sourceCol];

board[sourceRow][sourceCol] = nullptr;

board[destRow][destCol] = piece;

return true;

}

};

int main() {

ChessBoard board;

board.display();

cout << "Moving piece from a2 to a4\n";

board.movePiece("a2", "a4");

board.display();

cout << "Moving piece from a7 to a5\n";

board.movePiece("a7", "a5");

board.display();

cout << "Moving piece from b1 to c3\n";

board.movePiece("b1", "c3");

board.display();

return 0;

}

Question # 4:

#include <iostream>

#include <string>

using namespace std;

class RollerCoaster {

private:

string name;

int height;

int length;

int speed;

int capacity;

int currentNumRiders;

bool rideInProgress;

public:

// Default constructor

RollerCoaster() : name("Roller Coaster"), height(500), length(2000), speed(0), capacity(20), currentNumRiders(0), rideInProgress(false) {}

// Parameterized constructor

RollerCoaster(string \_name, int \_height, int \_length, int \_speed, int \_capacity)

: name(\_name), height(\_height), length(\_length), speed(\_speed), capacity(\_capacity), currentNumRiders(0), rideInProgress(false) {

// Verify if capacity is in multiples of two or three and greater than 3

if (capacity < 3)

capacity = 4;

else if (capacity % 2 != 0 && capacity % 3 != 0) {

capacity = (capacity + 1) / 2 \* 2; // Round up to the closest multiple of 2

}

}

// Getter functions

string getName() const {

return name;

}

int getHeight() const {

return height;

}

int getLength() const {

return length;

}

int getSpeed() const {

return speed;

}

int getCapacity() const {

return capacity;

}

int getCurrentNumRiders() const {

return currentNumRiders;

}

bool isRideInProgress() const {

return rideInProgress;

}

// Setter function for speed

void setSpeed(int \_speed) {

speed = \_speed;

}

// Load riders into roller coaster

int loadRiders(int numRiders) {

if (!rideInProgress && (currentNumRiders + numRiders) <= capacity) {

currentNumRiders += numRiders;

return 0; // All riders successfully seated

}

else {

return numRiders - (capacity - currentNumRiders); // Return the excess number of riders

}

}

// Start the ride

int startRide() {

if (!rideInProgress) {

if (currentNumRiders == capacity) {

rideInProgress = true;

return 0; // Ride started successfully

}

else {

return capacity - currentNumRiders; // Return the number of empty seats

}

}

else {

return -1; // Ride already in progress

}

}

// Stop the ride

void stopRide() {

rideInProgress = false;

}

// Unload riders from the roller coaster

void unloadRiders() {

if (!rideInProgress) {

currentNumRiders = 0;

}

}

// Accelerate the roller coaster

void accelerate() {

int lastDigit = 4;

speed += lastDigit;

}

// Apply brakes to slow down the roller coaster

void applyBrakes() {

int firstNonZeroDigit = 2;

speed -= firstNonZeroDigit;

}

};

int main() {

// Create roller coaster objects using both constructors

RollerCoaster defaultCoaster;

RollerCoaster customCoaster("Thunderbolt", 700, 2500, 0, 17);

// Demonstrate functionality with the custom coaster

cout << "Custom Roller Coaster Details:\n";

cout << "Name: " << customCoaster.getName() << "\n";

cout << "Height: " << customCoaster.getHeight() << " meters\n";

cout << "Length: " << customCoaster.getLength() << " meters\n";

cout << "Capacity: " << customCoaster.getCapacity() << " people\n\n";

cout << "Attempting to load 15 riders into the coaster...\n";

int excessRiders = customCoaster.loadRiders(15);

if (excessRiders == 0)

cout << "All riders successfully loaded.\n\n";

else

cout << "Only " << 15 - excessRiders << " riders loaded. " << excessRiders << " riders could not be accommodated.\n\n";

cout << "Starting the ride...\n";

int emptySeats = customCoaster.startRide();

if (emptySeats == 0)

cout << "Ride started successfully.\n\n";

else

cout << "Cannot start ride. " << emptySeats << " seats are empty.\n\n";

cout << "Attempting to load 5 more riders into the coaster...\n";

excessRiders = customCoaster.loadRiders(5);

if (excessRiders == 0)

cout << "All riders successfully loaded.\n\n";

else

cout << "Only " << 5 - excessRiders << " riders loaded. " << excessRiders << " riders could not be accommodated.\n\n";

cout << "Stopping the ride...\n";

customCoaster.stopRide();

cout << "Ride stopped.\n\n";

cout << "Unloading riders from the coaster...\n";

customCoaster.unloadRiders();

cout << "All riders unloaded.\n\n";

return 0;

}

Question#5:

#include <iostream>

#include <string>

#include <ctime>

using namespace std;

class BOGOCoupon {

private:

string coupon\_code;

int valid\_from;

int valid\_until;

string restaurant\_code;

int current\_date;

public:

BOGOCoupon(string code, time\_t from, time\_t until, string restaurant)

: coupon\_code(code), valid\_from(from), valid\_until(until), restaurant\_code(restaurant) {

current\_date=612;

}

BOGOCoupon() {

current\_date=612; //6th December

}

bool is\_valid() {

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

}

bool is\_valid\_for\_restaurant(string restaurant) {

return (restaurant\_code == restaurant);

}

string get\_coupon\_code() const {

return coupon\_code;

}

};

class Restaurant {

private:

string restaurant\_name;

string location;

string menu\_list;

string price\_list;

BOGOCoupon valid\_coupon\_codes\_list[5];

int Coupon\_Index=0;

static int coupons\_redeemed\_count;

public:

Restaurant(string name, string loc, string menu, string price)

: restaurant\_name(name), location(loc), menu\_list(menu), price\_list(price) {}

void display\_menu() {

cout << "Menu for " << restaurant\_name << " (" << location << "):\n";

cout << menu\_list << "\n";

}

float generate\_bill() {

cout << "Calculating bill...\n";

float total\_bill = 50.0;

cout << "Total Bill: $" << total\_bill << "\n";

return total\_bill;

}

void setCoupon(BOGOCoupon coupon)

{

valid\_coupon\_codes\_list[Coupon\_Index]=coupon;

Coupon\_Index++;

}

float apply\_discount(BOGOCoupon coupon) {

if (coupon.is\_valid() && coupon.is\_valid\_for\_restaurant(restaurant\_name)) {

coupons\_redeemed\_count++;

cout << "Coupon successfully applied!\n";

return generate\_bill();

} else {

cout << "Invalid or expired coupon for " << restaurant\_name << "\n";

return -1.0;

}

}

static int get\_coupons\_redeemed\_count() {

return coupons\_redeemed\_count;

}

};

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

class User {

private:

string name;

int age;

string mobile\_number;

BOGOCoupon coupons\_list[5];

int redeemed\_coupons\_list\_size;

public:

User(string n, int a, string mobile) : name(n), age(a), mobile\_number(mobile), redeemed\_coupons\_list\_size(0) {}

void accumulate\_coupon(BOGOCoupon coupon) {

coupons\_list[redeemed\_coupons\_list\_size++] = coupon;

cout << "Coupon added to the user's list!\n";

}

bool has\_valid\_coupon(string restaurant) {

for (int i = 0; i < redeemed\_coupons\_list\_size; ++i) {

if (coupons\_list[i].is\_valid() && coupons\_list[i].is\_valid\_for\_restaurant(restaurant)) {

return true;

}

}

return false;

}

float redeem\_coupon(Restaurant restaurant) {

cout << "Enter coupon code: ";

string coupon\_code;

cin >> coupon\_code;

for (int i = 0; i < redeemed\_coupons\_list\_size; ++i) {

if (coupons\_list[i].get\_coupon\_code() == coupon\_code) {

return restaurant.apply\_discount(coupons\_list[i]);

}

}

cout << "Coupon not found in the user's list.\n";

return -1.0;

}

};

int main() {

Restaurant foodHaven("FH", "City Center", "Sushi, Pad Thai, Mango Tango", "10.0, 12.0, 8.0");

Restaurant pixelBites("PB", "Cyber Street", "Binary Burger, Quantum Quinoa, Data Donuts", "15.0, 14.0, 5.0");

BOGOCoupon couponFoodHaven("FH-BOGO-12345", 412, 818, "FH"); //12April till 18 August

BOGOCoupon couponPixelBites("PB-BOGO-67890", 818, 1220, "PB"); //18August till 20 December

foodHaven.setCoupon(couponFoodHaven);

pixelBites.setCoupon(couponPixelBites);

foodHaven.display\_menu();

pixelBites.display\_menu();

User user("Ali", 25, "1234567890");

user.accumulate\_coupon(couponFoodHaven);

user.accumulate\_coupon(couponPixelBites);

cout << "Redeem coupon for Food Haven:\n";

user.redeem\_coupon(foodHaven);

cout << "Redeem coupon for Pixel Bites:\n";

user.redeem\_coupon(pixelBites);

cout << "Total coupons redeemed: " << Restaurant::get\_coupons\_redeemed\_count() << "\n";

return 0;

}