# **WEEK-3 DAY-1 TASK:**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class MenuItem {

public:

string name;

vector<MenuItem\*> subMenu;

MenuItem(const string& name) : name(name) {}

void addSubItem(MenuItem\* item) {

subMenu.push\_back(item);

}

};

class Menu {

private:

MenuItem\* currentMenu;

MenuItem\* rootMenu;

vector<MenuItem\*> menuStack;

int currentSelection;

public:

Menu(MenuItem\* root) : rootMenu(root), currentMenu(root), currentSelection(0) {}

void displayMenu() {

cout <<"\n -- " << currentMenu->name << " --" <<endl;

for (int i = 0; i < currentMenu->subMenu.size(); ++i) {

if (i == currentSelection)

cout << "-> ";

else

cout << " ";

cout << currentMenu->subMenu[i]->name << endl;

}

cout << "0: Exit 1: Down 2: Up 3: Enter 4: Back" <<endl;

}

void navigateDown() {

if (!currentMenu->subMenu.empty()) {

currentSelection = (currentSelection + 1) % currentMenu->subMenu.size();

}

}

void navigateUp() {

if (!currentMenu->subMenu.empty()) {

currentSelection = (currentSelection + currentMenu->subMenu.size() - 1) % currentMenu->subMenu.size();

}

}

void enter() {

if (!currentMenu->subMenu.empty()) {

menuStack.push\_back(currentMenu);

currentMenu = currentMenu->subMenu[currentSelection];

currentSelection = 0;

}

}

void back() {

if (!menuStack.empty()) {

currentMenu = menuStack.back();

menuStack.pop\_back();

currentSelection = 0;

}

}

void handleInput(int input) {

switch (input) {

case 1:

navigateDown();

break;

case 2:

navigateUp();

break;

case 3:

enter();

break;

case 4:

back();

break;

default:

cout << "Invalid input";

}

}

};

int main() {

MenuItem\* mainMenu = new MenuItem("MAIN MENU");

MenuItem\* settings = new MenuItem("Settings");

MenuItem\* media = new MenuItem("Media");

MenuItem\* displaySettings = new MenuItem("Display Settings");

MenuItem\* audioSettings = new MenuItem("Audio Settings");

MenuItem\* radio = new MenuItem("Radio");

MenuItem\* bluetoothAudio = new MenuItem("Bluetooth Audio");

mainMenu->addSubItem(settings);

mainMenu->addSubItem(media);

settings->addSubItem(displaySettings);

settings->addSubItem(audioSettings);

media->addSubItem(radio);

media->addSubItem(bluetoothAudio);

Menu menu(mainMenu);

int input;

while (true) {

menu.displayMenu();

cout << "Enter your choice: ";

cin >> input;

if (input == 0) break;

menu.handleInput(input);

}

return 0;

}

**OUTPUT:**

- MAIN MENU --

-> Settings

Media

0: Exit 1: Down 2: Up 3: Enter 4: Back

Enter your choice: 1

-- MAIN MENU --

Settings

-> Media

0: Exit 1: Down 2: Up 3: Enter 4: Back

Enter your choice: 2

-- MAIN MENU --

-> Settings

Media

0: Exit 1: Down 2: Up 3: Enter 4: Back

Enter your choice: 3

-- Settings --

-> Display Settings

Audio Settings

0: Exit 1: Down 2: Up 3: Enter 4: Back

Enter your choice: 3

-- Display Settings --

0: Exit 1: Down 2: Up 3: Enter 4: Back

Enter your choice: 4

-- Settings --

-> Display Settings

Audio Settings

0: Exit 1: Down 2: Up 3: Enter 4: Back

Enter your choice: 0

# **TASK-2**

#include <iostream>

#include <thread>

#include <mutex>

#include <chrono>

#include <cstdlib>

#include <ctime>

#include <random>

#ifdef \_WIN32

#include <windows.h>

#else

#include <unistd.h>

#endif

using namespace std;

class VehicleData {

private:

int speed;

int fuelLevel;

int temperature;

mutable mutex dataMutex;

// Using random device and Mersenne Twister engine for thread-safe random numbers

random\_device rd;

mt19937 gen{rd()};

public:

VehicleData() : speed(0), fuelLevel(100), temperature(70) {}

void updateData() {

lock\_guard<mutex> lock(dataMutex);

uniform\_int\_distribution<> speedDist(0, 120);

uniform\_int\_distribution<> fuelDist(0, 4); // Fuel decrease between 0 to 4

uniform\_int\_distribution<> tempDist(70, 110); // Temperature between 70 and 110 degrees

speed = speedDist(gen);

fuelLevel = max(0, fuelLevel - fuelDist(gen));

temperature = tempDist(gen);

}

int getSpeed() const {

lock\_guard<mutex> lock(dataMutex);

return speed;

}

int getFuelLevel() const {

lock\_guard<mutex> lock(dataMutex);

return fuelLevel;

}

int getTemperature() const {

lock\_guard<mutex> lock(dataMutex);

return temperature;

}

};

class Display {

public:

static void showData(const VehicleData& data) {

// Clear console screen depending on platform

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

int speed = data.getSpeed();

int fuel = data.getFuelLevel();

int temp = data.getTemperature();

cout << "================ Instrument Cluster ================" << endl;

cout << "Speed : " << speed << " km/h" << endl;

cout << "Fuel Level : " << fuel << "%" << endl;

cout << "Temperature : " << temp << "°C" << endl;

if (temp > 100)

cout << "WARNING: Engine overheating! " << endl;

if (fuel < 10)

cout << "WARNING: Low fuel level!" << endl;

cout << "===================================================" << endl;

}

};

void dataUpdater(VehicleData& data) {

while (true) {

data.updateData();

this\_thread::sleep\_for(chrono::seconds(1)); // Update every second

}

}

void dataDisplayer(VehicleData& data) {

while (true) {

Display::showData(data);

this\_thread::sleep\_for(chrono::seconds(2)); // Display every two seconds

}

}

int main() {

srand(static\_cast<unsigned int>(time(0))); // Seed random number generator

VehicleData vehicleData;

// Start updater and displayer threads

thread updater(dataUpdater, ref(vehicleData));

thread displayer(dataDisplayer, ref(vehicleData));

// Join threads so main thread waits for both to finish

updater.join();

displayer.join();

return 0;

}

**OUTPUT:**

================ Instrument Cluster ================

Speed : 47 km/h

Fuel Level : 0%

Temperature : 89°C

WARNING: Low fuel level!

==================================================

# **TASK-3:**

#include <iostream>

#include <queue>

#include <string>

#include <cstdlib>

#include <ctime>

#include <thread>

#include <chrono>

using namespace std;

class Event {

public:

string eventType;

int x, y;

int xEnd, yEnd;

string timestamp;

Event(const string& type, int x, int y, const string& time, int xEnd = 0, int yEnd = 0)

: eventType(type), x(x), y(y), xEnd(xEnd), yEnd(yEnd), timestamp(time) {}

};

string generateTimestamp() {

time\_t now = time(0);

tm\* localTime = localtime(&now);

char buffer[20];

strftime(buffer, sizeof(buffer), "%H:%M:%S", localTime);

return string(buffer);

}

queue<Event> eventQueue;

void generateEvents() {

while (true) {

string eventType = (rand() % 2 == 0) ? "Tap" : "Swipe";

int x = rand() % 500;

int y = rand() % 500;

string timestamp = generateTimestamp();

if (eventType == "Tap") {

eventQueue.push(Event("Tap", x, y, timestamp));

} else {

int xEnd = x + (rand() % 201 - 100);

int yEnd = y + (rand() % 201 - 100);

eventQueue.push(Event("Swipe", x, y, timestamp, xEnd, yEnd));

}

this\_thread::sleep\_for(chrono::milliseconds(1000));

}

}

void processEvents() {

while (true) {

if (!eventQueue.empty()) {

Event currentEvent = eventQueue.front();

eventQueue.pop();

cout << "Event at " << currentEvent.timestamp << ": ";

if (currentEvent.eventType == "Tap") {

cout << "Tap at (" << currentEvent.x << ", " << currentEvent.y << ")" << endl;

} else if (currentEvent.eventType == "Swipe") {

int dx = currentEvent.xEnd - currentEvent.x;

int dy = currentEvent.yEnd - currentEvent.y;

string direction;

if (abs(dx) > abs(dy)) {

direction = (dx > 0) ? "Right" : "Left";

} else {

direction = (dy > 0) ? "Down" : "Up";

}

cout << "Swipe from (" << currentEvent.x << ", " << currentEvent.y << ") "

<< "to (" << currentEvent.xEnd << ", " << currentEvent.yEnd << ") - "

<< "Direction: " << direction << endl;

}

this\_thread::sleep\_for(chrono::milliseconds(1000));

}

}

}

int main() {

srand(static\_cast<unsigned int>(time(0)));

thread eventGenerator(generateEvents);

thread eventProcessor(processEvents);

eventGenerator.join();

eventProcessor.join();

return 0;

}

**OUTPUT:**

Event at 05:24:17: Swipe from (68, 208) to (-23, 277) - Direction: Left

Event at 05:24:18: Swipe from (142, 182) to (73, 178) - Direction: Left

Event at 05:24:19: Swipe from (274, 476) to (279, 401) - Direction: Up

Event at 05:24:20: Swipe from (152, 214) to (94, 201) - Direction: Left

Event at 05:24:21: Swipe from (312, 471) to (399, 431) - Direction: Right

Event at 05:24:22: Swipe from (485, 314) to (454, 354) - Direction: Down

# **TASK-4:**

#include <iostream>

#include <map>

#include <string>

using namespace std;

class Theme {

public:

string backgroundColor;

string fontColor;

int fontSize;

string iconStyle;

Theme() : backgroundColor("DefaultBG"), fontColor("DefaultFont"), fontSize(12), iconStyle("DefaultIcon") {}

Theme(const string& bg, const string& font, int size, const string& icon)

: backgroundColor(bg), fontColor(font), fontSize(size), iconStyle(icon) {}

void apply() const {

cout << "Applying Theme: " << backgroundColor << ", " << fontColor << ", "

<< fontSize << "px Font Size, " << iconStyle << " Icons." << endl;

}

void display() const {

cout << "Theme Preview: " << backgroundColor << " Background, " << fontColor

<< " Font, " << fontSize << "px Font Size, " << iconStyle << " Icons." << endl;

}

};

int main() {

map<string, Theme> themes;

themes["classic"] = Theme("White", "Black", 14, "Square");

themes["sport"] = Theme("Red", "White", 16, "Bold");

themes["eco"] = Theme("Green", "Brown", 12, "Leaf");

string choice;

while (true) {

cout << "\nAvailable Themes: classic, sport, eco, exit" << endl;

cout << "Enter theme name to apply: ";

cin >> choice;

if (choice == "exit" ) {

break;

}

auto it = themes.find(choice);

if (it != themes.end()) {

cout << "\n" << choice << " Theme selected: " << endl;

it->second.display();

it->second.apply();

} else {

cout << "Theme not found. Please try again." << endl;

}

}

return 0;

}

//this is fourth task

**OUTPUT:**

Available Themes: classic, sport, eco, exit

Enter theme name to apply: eco

eco Theme selected:

Theme Preview: Green Background, Brown Font, 12px Font Size, Leaf Icons.

Applying Theme: Green, Brown, 12px Font Size, Leaf Icons.

Available Themes: classic, sport, eco, exit

Enter theme name to apply: exit