#include <iostream>

#include <iomanip>

using namespace std;

// Structure for Appointment

struct Appointment {

int startTime; // Start time in minutes from 00:00

int endTime; // End time in minutes from 00:00

Appointment\* next;

Appointment(int start, int end) : startTime(start), endTime(end), next(nullptr) {}

};

// Class for Appointment Schedule

class Schedule {

private:

Appointment\* head;

const int minDuration = 30; // Minimum duration for an appointment

const int maxDuration = 120; // Maximum duration for an appointment

public:

Schedule() : head(nullptr) {}

// Utility function to convert time in minutes to HH:MM format

string formatTime(int minutes) {

int hours = minutes / 60;

int mins = minutes % 60;

return (hours < 10 ? "0" : "") + to\_string(hours) + ":" + (mins < 10 ? "0" : "") + to\_string(mins);

}

// Display free slots

void displayFreeSlots(int startDay = 0, int endDay = 1440) {

Appointment\* temp = head;

int freeStart = startDay;

while (temp != nullptr) {

if (temp->startTime > freeStart) {

cout << "Free Slot: " << formatTime(freeStart) << " - " << formatTime(temp->startTime) << endl;

}

freeStart = max(freeStart, temp->endTime);

temp = temp->next;

}

if (freeStart < endDay) {

cout << "Free Slot: " << formatTime(freeStart) << " - " << formatTime(endDay) << endl;

}

}

// Book an appointment

bool bookAppointment(int start, int end) {

if (end <= start || (end - start) < minDuration || (end - start) > maxDuration) {

cout << "Invalid appointment duration.\n";

return false;

}

Appointment\* newAppointment = new Appointment(start, end);

if (head == nullptr || start < head->startTime) {

if (head != nullptr && end > head->startTime) {

cout << "Time slot overlaps with an existing appointment.\n";

delete newAppointment;

return false;

}

newAppointment->next = head;

head = newAppointment;

return true;

}

Appointment\* temp = head;

while (temp->next != nullptr && temp->next->startTime < start) {

temp = temp->next;

}

if (temp->endTime > start || (temp->next != nullptr && temp->next->startTime < end)) {

cout << "Time slot overlaps with an existing appointment.\n";

delete newAppointment;

return false;

}

newAppointment->next = temp->next;

temp->next = newAppointment;

return true;

}

// Cancel an appointment

bool cancelAppointment(int start) {

if (head == nullptr) {

cout << "No appointments to cancel.\n";

return false;

}

if (head->startTime == start) {

Appointment\* temp = head;

head = head->next;

delete temp;

return true;

}

Appointment\* temp = head;

while (temp->next != nullptr && temp->next->startTime != start) {

temp = temp->next;

}

if (temp->next == nullptr) {

cout << "Appointment not found.\n";

return false;

}

Appointment\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

return true;

}

// Sort appointments based on time using pointer manipulation

void sortAppointments() {

if (head == nullptr || head->next == nullptr)

return;

Appointment\* sorted = nullptr;

while (head != nullptr) {

Appointment\* current = head;

head = head->next;

if (sorted == nullptr || current->startTime < sorted->startTime) {

current->next = sorted;

sorted = current;

} else {

Appointment\* temp = sorted;

while (temp->next != nullptr && temp->next->startTime < current->startTime) {

temp = temp->next;

}

current->next = temp->next;

temp->next = current;

}

}

head = sorted;

}

// Display all appointments

void displayAppointments() {

Appointment\* temp = head;

if (temp == nullptr) {

cout << "No appointments scheduled.\n";

return;

}

cout << "Scheduled Appointments:\n";

while (temp != nullptr) {

cout << formatTime(temp->startTime) << " - " << formatTime(temp->endTime) << endl;

temp = temp->next;

}

}

// Destructor to free memory

~Schedule() {

while (head != nullptr) {

Appointment\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

Schedule schedule;

int choice;

do {

cout << "\nAppointment Schedule Menu:\n";

cout << "1. Display Free Slots\n";

cout << "2. Book Appointment\n";

cout << "3. Cancel Appointment\n";

cout << "4. Display Appointments\n";

cout << "5. Sort Appointments\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

schedule.displayFreeSlots();

break;

case 2: {

int start, end;

cout << "Enter start time (in minutes from 00:00): ";

cin >> start;

cout << "Enter end time (in minutes from 00:00): ";

cin >> end;

if (schedule.bookAppointment(start, end)) {

cout << "Appointment booked successfully.\n";

}

break;

}

case 3: {

int start;

cout << "Enter start time of the appointment to cancel (in minutes from 00:00): ";

cin >> start;

if (schedule.cancelAppointment(start)) {

cout << "Appointment canceled successfully.\n";

}

break;

}

case 4:

schedule.displayAppointments();

break;

case 5:

schedule.sortAppointments();

cout << "Appointments sorted successfully.\n";

break;

case 6:

cout << "Exiting the program.\n";

break;

default:

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

}

} while (choice != 6);

return 0;

}