```
#include <iostream>

#include <string>


using namespace std;


//
====================================================================================
====================================================================================
================

// Class declarations




class Citizen

{

	int citizen_id; // Personal details (data)

	string name;

	int age;

	char sex;

	bool comorbidities;


	string slot_date;


	// Implementation details

	int priority; // For the calculation details, see the get_priority function of this class

	string password;

	bool demoted; // True if the citizen has been relegated to the end of the queue-list for
missing their alloted slot.

	Citizen *next = nullptr;


public:

	// Visibility flags - Modified only by running allotment process

	bool visible;	// True when citizen can see their slot
```

```
        bool slot_today; // True when the slot allocated is today

        Citizen(string nm, int cit_id, char sx, int ag, string pswd)
        {
                /*
                 * Constructor
                 */
                citizen_id = cit_id;
                name = nm;
                age = ag;
                sex = sx;
                slot_date = "Not Alloted Yet.";

                visible = false;
                slot_today = false;

                password = pswd;
                priority = (visible * 10000) + (comorbidities * 1000) + age;
                demoted = false;
        }

        int get_priority();
        bool is_pswd(string pswd);
        string get_details(string pswd);

        friend class Admin; // class Admin in "Admin.h" header
        friend class CitizenQueueList;
};

class CitizenQueueList
{
        Citizen *rear;
        Citizen *front;
        void display_top_n(unsigned n);
        void display_today();
        void remove(Citizen *user, string pswd);
        Citizen *dequeue();

public:
        CitizenQueueList()
        {
                /*
         * Constructor
         */
                front = nullptr;
                rear = nullptr;
        }

        void enqueue(Citizen *p);
        Citizen *search(int citizenID);
        int get_min_priority();

        friend class Admin;
};

//
==========================================================================================
==========================================================================================
==========================

// Citizen implementation
int Citizen::get_priority()
{
	/*
	 * PRIORITY CALCULATION :
	 * The rules of priority calculation are :
	 * 1.If a citizen has been allocated a slot, they cannot be overriden
	 *   by someone who has not been allocated a slot.
	 * 2.Citizens with comorbidities have a higher priority than those without.
	 * 3.Older citizens have higher priority.
	 * 4.Demoted citizens' priorities depend on the priority of the last person in the queue-list
	 *          when they were demoted and on their 'visible' member.
	 *
	 * This function updates and returns the priority of a citizen.
	 *
	 */
	if (!demoted)
		priority = (visible * 10000) + (comorbidities * 1000) + age;
	else
		priority = (visible * 10000) + priority;
	return priority;
}

bool Citizen::is_pswd(string pswd)
{
	/*
	 * Verifies the citizen's password.
	 * Returns true if the password matches.
	 */
	return password == pswd;
}

string Citizen::get_details(string pswd)
{
	/*
	 * Returns string representation of the citizen's details.
	 * Protected by password.
	 */
	string details;
	if (is_pswd(pswd))
	{
		details += "\n\n\tCitizenID : ";
		details += to_string(citizen_id);
		details += "\n\n\t";
		details += "Name : ";
		details += name;
		details += "\n\n\t";
		details += "Age : ";
		details += to_string(age);
		details += "\n\n\t";
		details += "Sex : ";
		details += sex;
		details += "\n\n\t";
		details += "Slot Date : ";
		details += slot_date;
		details += "\n\n\t";
		return details;
	}
	return "Access Denied.";
}

//
================================================================================
================================================================================
==========
// CitizenQueueList implementation
```

```
Citizen *CitizenQueueList::search(int citizenID)
{
        /*
         *  Returns the pointer to the citizen if found,
         *  otherwise returns null.
         */
        Citizen *cur = front;
        while (cur != nullptr)
        {
                if (cur->citizen_id == citizenID)
                        break;
                cur = cur->next;
        }
        return cur;
}

void CitizenQueueList::enqueue(Citizen *new_citizen)
{
        /*
         * Adds citizen to the queue-list as per priority.
         */
        if (front == nullptr) //First registration
        {
                front = new_citizen;
                rear = new_citizen;
        }
        else
        {
                Citizen *cur = front;
                Citizen *prev = nullptr;
```

```
                    while (cur != nullptr && cur->get_priority() >= new_citizen->get_priority()) // Finding
the location for insertion
                    {
                              prev = cur;
                              cur = cur->next;
                    }
                    if (prev == nullptr) // Appending to the head
                    {
                              new_citizen->next = front;
                              front = new_citizen;
                    }
                    else if (cur == nullptr) //Appending to the rear
                    {
                              prev->next = new_citizen;
                              rear = new_citizen;
                    }
                    else // Appending in the middle
                    {
                              prev->next = new_citizen;
                              new_citizen->next = cur;
                    }
          }
}

Citizen *CitizenQueueList::dequeue()
{
          /*
           * Removes and returns the front of the queue-list
           */
          Citizen *top = front;
```

```
	if (front == rear)
	{
		front = nullptr;
		rear = nullptr;
	}
	else
	{
		front = front->next;
		top->next = nullptr;
	}
	return top;
}

void CitizenQueueList::remove(Citizen *user, string pswd)
{
	/*
	 * Allows the removal of a person from the queue-list.
	 * Can only be accessed with the person's consent.
	 */
	if (!user->is_pswd(pswd)) // Prevents malicious calls of this function from affecting the
queue-list, enforcing the person's consent.
		return;

	Citizen *cur = front;
	Citizen *prev = nullptr;
	while (cur->citizen_id != user->citizen_id && cur != nullptr)
	{ // Finding the person in the queue-list
		prev = cur;
		cur = cur->next;
	}
	if (prev == nullptr) // Removing from the front
```

```
                front = cur->next;
        else // Removing from other positions
                prev->next = cur->next;
        if (rear == cur) // Removing from rear
                rear = prev;
        cur->next = nullptr;
        delete (cur);
}

void CitizenQueueList::display_top_n(unsigned n)
{
        /*
         * Displays CitizenID, Name and Slot Details
         * of the top n entries of the queue-list if they exist.
         */
        Citizen *user = front;
        unsigned count;
        for (count = 1; count <= n && user != nullptr; count++)
        {
                cout << "\n\t" << count << ")\t CitizenID: " << user->citizen_id << "\n\t\t Name:" <<
user->name << "\n\t\t Slot Date:" << user->slot_date << endl;
                user = user->next;
        }
        if (count - 1 < n)
                cout << "Only " << (count - 1) << " entries in queue-list" << endl;
}

void CitizenQueueList::display_today()
{
        /*
         * Displays CitizenID, Name and Slot Details of
```

```
             * citizens with today's slot.
             */
            Citizen *user = front;
            int count = 1;
            while (user != NULL && user->slot_today)
            {
                        cout << "\n\t" << count << ")\t CitizenID:" << user->citizen_id << "\n\t\t Name:" <<
user->name << "\n\t\t Slot Date:" << user->slot_date << endl;
                        user = user->next;
                        count++;
            }
            if (count == 1)
                        cout << "\n\tNo citizens in today's slot." << endl;
}

int CitizenQueueList::get_min_priority()
{
            /*
             * Returns the priority of the rear element (the lowest priority).
             * Useful for moving citizens to the bottom of the queue-list.
             * Used in allotment process in Admin class.
             */
            return rear->get_priority();
}

//
==================================================================================
==================================================================================
========================
// DATE allotment for the vaccine drive

bool isLeap(int yr)
{
        /*
         * Returns true if the year is a leap year
         */
        return yr % 4 == 0 && yr % 100 != 0;
}
string date_string(int day, int month, int year)
{
        /*
         * Returns the string form of a date
         */
        return to_string(day) + "/" + to_string(month) + "/" + to_string(year);
}
string get_next_date(int day, int month, int year)
{
        /*
         * Returns the next date in string form, given input
         * today's day, month and year
         */
        int new_day, new_month, new_year;
        enum months
        {
                Jan = 1,
                Feb,
                Mar,
                Apr,
                May,
                Jun,
                Jul,
                Aug,
                Sep,
                Oct,
                Nov,
                Dec
        };
        bool ordinary = false;
        new_year = year;
        switch (day)
        {
        case 28:
                if (month == Feb && !isLeap(year)) // 28th Feb, non leap year
                {
                        new_month = Mar;
                        new_day = 1;
                }
                else
                        ordinary = true;
                break;
        case 29:
                if (month == Feb) // 29th Feb, leap year
                {
                        new_month = Mar;
                        new_day = 1;
                }
                else
                        ordinary = true;
                break;
        case 30:
                if (month == Apr || month == Jun || month == Sep || month == Nov) // 30 day
        months
                {
                    new_month = month + 1;
                    new_day = 1;
                }
                else
                    ordinary = true;
                break;
            case 31:
                if (month == Dec) // 31st Dec
                {
                    new_month = Jan;
                    new_year++;
                    new_day = 1;
                }
                else // Other 31 day months
                {
                    new_month = month + 1;
                    new_day = 1;
                }
                break;
            default:
                ordinary = true;
                break;
            }

            if (ordinary) // Month does not change; day is incremented by 1
            {
                new_month = month;
                new_day = day + 1;
            }

            return date_string(new_day, new_month, new_year);
        }


        //
        ====================================================================================
        ====================================================================================
        =====


        class Admin
        {
            string pswd;
            static int available_shots; // The number of vaccine shots available per day
            static string last_date;    // Used by the run_process function of this class to record the date
        of its last execution

        public:
            static int age_eligibility; // The minimum age eligible for registration

            Admin()
            {
                /*
                 * Constructor
                 */
                pswd = "vaccine";
            }
            bool run_process(CitizenQueueList &cqueue);
            void display_top_n(CitizenQueueList &cqueue);
            void display_today(CitizenQueueList &cqueue);
```

```
	void change_eligibility(CitizenQueueList &cqueue);
	void change_available_shots();
	bool verify_password(string pass);
	void remove(Citizen *user, string pswd, CitizenQueueList &cqueue);
	void remove_age_ineligible(CitizenQueueList &cqueue);

	friend class CitizenQueueList;
};
//
=================================================================================
===================
// Admin class implementation

// Static member initializations
int Admin::age_eligibility = 18;
int Admin::available_shots = 1;
string Admin::last_date = "";

bool Admin::verify_password(string pass)
{
	/*
	 * Verifies the Admin password.
	 * Returns true if the password matches.
	 */
	return pass == pswd;
}

bool Admin::run_process(CitizenQueueList &cqueue)
{
	/*
	 * The most important functionality of this class.
	 * It allocates slots to citizens in a priority queue-list.
 * Returns true if the allocation process has executed.
 */

Citizen *selected = cqueue.front;
Citizen *dq = nullptr;

// Extracting today's date
time_t now = time(0);
tm *ltm = localtime(&now);
int day = ltm->tm_mday;
int month = ltm->tm_mon + 1;
int year = ltm->tm_year + 1900;

/*
 * This code has to be uncommented to generate full output, as
 * otherwise, the program will have to remain running over several days
 * to demonstrate the full functionality, due to use of <ctime>
 */
/*
 cout << "Enter day :";
 cin >> day;
 cout << "Enter month :";
 cin >> month;
 cout << "Enter year :";
 cin >> year;
 */

string today = date_string(day, month, year); // The representation of a today's date as a
string

if (today == last_date) // This prevents multiple allotment rounds on the same day.
```

```
                return false;

        // ALLOTMENT PROCESS:

        // For citizens who have been allocated slots previously
        while (selected != nullptr && selected->visible)
        {
                if (selected->slot_date == today) // Citizens whose slot is today
                {
                        selected->slot_today = true;  // Modifies the citizen's slot_today member
                        selected = selected->next;
                }
                else
                {
                        // Citizens whose slot occurred yesterday, but who failed to get vaccinated,
                        // need to be removed to the end of the queue-list (demoted) so they can be
assigned fresh slots.

                        selected = selected->next; // Preserving the traversal before moving the
element

                        int new_priority = cqueue.get_min_priority() - 1;
                        dq = cqueue.dequeue();
                        dq->visible = false;
                        dq->slot_today = false;
                        dq->demoted = true;
                        dq->priority = new_priority;
                        dq->slot_date = "Not Alloted Yet.";

                        cqueue.enqueue(dq);
                }
```

```
	}
	string dslot = get_next_date(day, month, year); // Generates the string representation for the following day.

	// Allocates the following day's slot to the citizens and enables them to check their date slots.
	// Only a maximum of 'available_shots' number of citizens can be allocated a slot on a given day.
	for (int count = 1; count <= available_shots; count++)
	{
		if (selected == nullptr)
			break;
		selected->slot_date = dslot;
		selected->visible = true;
		selected = selected->next;
	}

	last_date = today; // Modifies the last_date member to today's date

	return true;
}

void Admin::display_top_n(CitizenQueueList &cqueue)
{
	unsigned n;
	cout << "\n\tHow many entries are to be displayed ?";
	cin >> n;
	cqueue.display_top_n(n);
}

void Admin::display_today(CitizenQueueList &cqueue)
{
        cqueue.display_today();
}
void Admin::change_eligibility(CitizenQueueList &cqueue)
{
        /*
         * Allows admin to change the minimum eligibility age
         */
        cout << "\n\tEnter eligibility age : ";
        cin >> age_eligibility;
        remove_age_ineligible(cqueue);
        cout << "\n\t\tChanged successfully." << endl;
}
void Admin::change_available_shots()
{
        /*
         * Allows admin to change available_shots as per actual availability
         */
        cout << "\n\tEnter new number : ";
        cin >> available_shots;
        cout << "Changed successfully." << endl;
}

void Admin::remove(Citizen *user, string pswd, CitizenQueueList &cqueue)
{
        cqueue.remove(user, pswd);
}

void Admin::remove_age_ineligible(CitizenQueueList& cqueue)
{
        Citizen* cur = cqueue.front;
        Citizen* prev = nullptr;
        Citizen* to_delete = nullptr;
        bool delete_flag = false;

        while(cur != nullptr){
                delete_flag = cur->age < age_eligibility && !cur->visible;
                if (delete_flag)
                {
                        if (prev == nullptr) // Removing from the front
                                cqueue.front = cur->next;
                        else // Removing from other positions
                                prev->next = cur->next;
                        if (cqueue.rear == cur) // Removing from rear
                                cqueue.rear = prev;
                        to_delete = cur;
                        cur = cur->next;
                        to_delete->next = nullptr;
                        delete(to_delete);
                }
                else
                {
                        prev = cur; // change prev only if cur is not to be deleted
                        cur = cur->next;
                }

        }
}
//
================================================================================
================================================================================
===========

//USER INTERFACE(Login, Registeration, admin login);
```

```
CitizenQueueList cqueue; // global - The queue-list object which holds all the data of registered
citizens

// Function declarations

void registration();

void user_login();

void admin_login();


int main()

{

        int c= -1;

        do

        {

                cout << "\n\t\t\t\t********************************************";

    cout << "\n\t\t\t\t*    COVID19 VACCINE MANAGEMENT SYSTEM    *";

    cout << "\n\t\t\t\t********************************************";

                cout << "\n\t -->> MAIN MENU <<--" << endl; // MAIN MENU

                cout << "\n\t1. Register for vaccine." << endl;

                cout << "\n\t2. Login as user." << endl;

                cout << "\n\t3. Login as Admin." << endl;

                cout << "\n\tEnter 0 to exit." << endl;

                cout << "\n\t----------------------" << endl;

                cout << "\n\t Enter choice :";

                cin >> c;

                switch (c)

                {

                case 1:

                        registration();

                        break;

                case 2:
```

```
                user_login();
                break;
            case 3:
                admin_login();
                break;
            case 0:
                cout << "\n\t\tThank you for using App." << endl;
                break;
            default:
              system("cls");
                cout << "\n\t\t Invalid Choice... Please Try Again....," << endl;
                cout << "\n\t\t Press Any Key To Continue: " << endl;
                main();
            }
        } while (c);

        return 0;
}
//
==============================================================================
============

void registration()
{
        string name;
        int age, id;
        char ch, sex;
        cout << "\n\t Enter your name: ";
        cin >> name;
        cout << "\n\t Enter citizen ID: ";
        cin >> id;
```

```
Citizen *found = cqueue.search(id);
if (found != nullptr)
{
        cout << "\n\t\tCitizen with this citizen id already registered ! " << endl;
        return;
}

// New registration details
do // Validation
{
        cout << "\n\t Enter Sex: (m/f)";
        cin >> sex;
        if (sex != 'm' && sex != 'f')
                cout << "\n\t\tInvalid Input!";
        else
                break;
} while (true);

cout << "\n\t Enter your age: ";
cin >> age;
if (Admin::age_eligibility > age)
{
        cout << "\n\t\tSorry Cannot Register! You are underage! ";
        return;
}

string pswd, pswd2;
do
{ // Password validation
        cout << "\n\t Set your password: ";
        cin >> pswd;
```

```
                cout << "\n\t Confirm password: ";
                cin >> pswd2;
                if (pswd != pswd2)
                        cout << "\n\t Password does not match! Set up your password again!";
        } while (pswd != pswd2);

        Citizen *new_citizen = new Citizen(name, id, sex, age, pswd);
        cqueue.enqueue(new_citizen);
        cout << "\n\t\tRegistered successfully, please log in to see your details." << endl;
}

//
=================================================================================
============

void user_login()
{
        /*
         *  This function is responsible for the
         *  login experience of a regular user (citizen)
         */
        void admin_authorise(Citizen * user, string pass); // Function which is called when Admin
authorisation is required

        int citizenID;

        system("cls");
        cout << "\n\t Enter your Citizen ID :";
        cin >> citizenID;
        Citizen *user = cqueue.search(citizenID);

        if (user == nullptr) // User no1t found
```

```
{
    cout << "\n\t\t Sorry, please register first or" << endl;
    cout << "\n\t\t enter a valid Citizen ID." << endl;
    return;
}
else
{
    int count = 3; // Attempts to enter the correct password
    string pswd;
    while (count != 0)
    {
        cout << "\n\t Enter your password :";
        cin >> pswd;
        if (!user->is_pswd(pswd)) // Checking the entered password
        {
            count--; // Incorrect attempt reduces the count
            cout << "\n\t\tIncorrect password." << endl;
            cout << "\n\t\t" << count << " attempts remaining." << endl;
        }
        else
            break;
    }
    if (count == 0) // Failed to enter the correct password
        return;


    cout << "\n\tLogged in successfully." << endl;

    // Display user details
    cout << "\n\t -->> CITIZEN DETAILS <<--";
    cout << user->get_details(pswd) << endl;
```

```
		cout << "----------------------" << endl;
		// User Menu
		cout << "\t USER MENU :" << endl;
		if (user->slot_today) // Facility available only on vaccination day
			cout << "\n\t1. Confirm vaccination completion. " << endl;
		cout << "\n\tEnter 0 to logout." << endl;
		cout << "----------------------" << endl;
		bool vaccinated = false; // Used for terminating the user login after completion of
vaccination
		int choice = -1;
		do
		{
			cout << "\n\t Enter choice :";
			cin >> choice;
			switch (choice)
			{

			case 1:
				if (user->slot_today)
				{
					cout << "\n\t\tPlease provide authorisation for confirming
the vaccination" << endl; // This step requires admin authorisation
					admin_authorise(user, pswd);
					vaccinated = true;
				}
				else
					cout << "\n\t\tPlease enter a valid choice." << endl;
				break;
			case 0:
				cout << "\n\t\tSigned out." << endl;
				return;
			default:
```

```
                        cout << "\n\t\tPlease enter a valid choice." << endl;
                    }
                } while (choice != 0 && !vaccinated);
            }
}

void admin_authorise(Citizen *user, string pass)
{
            /*
             * Helper to the user_login function, it enables
             * Admin authorisation for de-registration of the user.
             * Note: The parameter 'string pass' is required by the Admin::remove() function
             * which is used here.
             */
            Admin admin;
            int count = 3; // Attempts to enter the correct password
            string admin_pswd;
            while (count != 0)
            {
                cout << "\n\tEnter admin password :";
                cin >> admin_pswd;
                if (!admin.verify_password(admin_pswd))
                {
                    count--; // Count decremented after incorrect attempt.
                    cout << "\n\t\tIncorrect password." << endl;
                    cout << "\n\t " << count << " attempts remaining." << endl;
                }
                else
                    break;
            }
            if (count == 0) // Admin authorisation not provided
```

```
        {
                cout << "\n\tSorry, you are not authorised." << endl;
                return;
        }
        admin.remove(user, pass, cqueue); // De-registration of the user
        cout << "\n\tCongratulations on getting vaccinated !" << endl;
        cout << "\n\tYou've been signed out and your account has been removed." << endl;
        cout << "\n\tThank you for using the App" << endl;
}

//
===============================================================================
=============

void admin_login()
{
        /*
         * Provides interface for Admin to execute the vaccination allotment process
         * and some other administrative tasks.
         */

        void display_handler(Admin & ad); // Function declaration - used in handling the display of
registrations

        Admin admin;
        int count = 3; // Attepmts to enter the correct password
        string pswd;
        while (count != 0)
        {
                cout << "\n\tEnter your password :";
                cin >> pswd;
                if (!admin.verify_password(pswd))
```

```
		{
			count--; // Decrement attempts after incorrect entry
			cout << "\n\tIncorrect password!";
			cout << "\n\t" << count << " attempts remaining." << endl;
		}
		else
			break;
	}
	if (count == 0) // Login authorisation not provided
		return;
	cout << "\n\t Logged in successfully! " << endl;

	bool proc_executed;
	int c;
	do
	{
		// Admin Menu
		cout << "\n\t--------------------------";
		cout << "\n\t -->> ADMIN MENU <<-- " << endl;
		cout << "\n\t1.Execute time allotment process" << endl;
		cout << "\n\t2.Display registrations." << endl;
		cout << "\n\t3.Change number of available shots." << endl;
		cout << "\n\t4.Change minimum age requirement for vaccination" << endl;
		cout << "\n\tEnter 0 to logout." << endl;
		c = -1;
		cout << "\n\tEnter your choice: ";
		cin >> c;
		switch (c)
		{
		case 1:
			proc_executed = admin.run_process(cqueue); // Refer to Admin class
```

```
            if (!proc_executed)
                cout << "\n\tCannot run allotment multiple times on the same day." << endl;
            else
                cout << "\n\t\t*** Allotment complete ***" << endl;
            break;
        case 2:
            display_handler(admin);
            break;
        case 3:
            admin.change_available_shots();
            break;
        case 4:
            admin.change_eligibility(cqueue);
            break;
        case 0:
            cout << "\n\t Signed out!" << endl;
            return;
        default:
            cout << "\n\tPlease enter a valid choice." << endl;
        }
    } while (c != 0);
}

void display_handler(Admin &ad)
{
    int c = -1;
    do
    {
        cout << "\n\t---------------------------";
        cout << "\n\t Display menu" << endl;
```

```
            cout << "\n\tEnter 1 for displaying n entries" << endl;
            cout << "\n\tEnter 2 for displaying citizens in today's slot" << endl;
            cout << "\n\tEnter 0 to return to main Admin Menu." << endl;
            cout << "\n\tEnter choice :";
            cin >> c;
            switch (c)
            {
            case 1:
                ad.display_top_n(cqueue);
                break;
            case 2:
                ad.display_today(cqueue);
                break;
            case 0:
                cout << "\n\tReturning to main Admin menu ..." << endl;
                break;
            default:
                cout << "\n\tEnter a valid input.";
                break;
            }
        } while (c != 0);
}

//
==============================================================================
============
```