Programming and Data Structures (CS221): Mini Project Quarantine Management System

Shubhankar Pandit: 18-13-104

Electrical Engineering, 4th Semester

July 11, 2020

Abstract

This report discusses the code design, functionalities provided to the user, the code layout, the features and all the shortcomings of the program created to manage a hypothetical quarantine facility in NIT Silchar.

Contents

1 Introduction

1.1 Problem Statement

According to the given problem statement, a quarantine facility has been set up at NIT Silchar, which can accommodate a maximum of 500 people. The building being used for quarantine is multi-storied, consisting of

- Ground floor with 200 rooms
- First floor with 150 rooms
- Second floor with 150 rooms

It has been decided that the floors will be allotted according to the age of the patient, with the oldest patients getting the lowest floors. The distribution is given as follows

- Ground Floor - above 60 years of age
- First floor - b/w 40 years to 60 years
- Second floor - below 40 years

The nodal officer who administers the quarantine centre has to keep track of the records of the quarantined persons like the name of the person, address, age, arrival date, coming from, going to, allotted room no, discharged date etc.
In this project, we attempt to solve this practical problem by using a data container to store the respective patient's record; giving the user the ability to search any patient's records using their name or room number; the ability to sort the records with respect to their allotted room numbers; and the feature to display the list of occupied and vacant rooms.

1.2 Assumptions

Certain assumptions have been made in the process of making this project

1. It has been assumed that any number of patients can be inputted into the database, but only a maximum of 500 patients will be allotted a room.
2. Patients can be discharged either all at once, or individually. Discharged date is only updated when a discharge process is carried out.
3. Patients can be in 3 possible states, *i.e.* their record exists inside the database but they haven't been allotted a room; they are quarantined inside a room; they have been discharged and currently do not occupy a room.
4. The room details are defined by two parameters, the room number and the floor number. 0 signifies ground floor, 1 first floor, and 2 second floor.
5. Rooms are numbered from 1 to 200 for the ground floor and 1 to 150 for the first and second floors.

2 Code Design

Class has been used as a data container with various attributes and methods to keep the entire program as abstract as possible. A total of two classes were created, one pertaining to the patient, and one to the entire facility. Attributes and methods specific to a patient have been defined in the class `patient` and all other general attributes and methods in the class `quarantine`.
The class `quarantine` makes use of a *vector* of class `patient`. Since *vectors* in *C++* are a form of dynamic arrays which can resize themselves as needed, this *vector* of `patient` is conveniently used inside class *quarantine* to add and remove patients freely.

Inside `main()`, a `switch` case has been used to carry out the request of the user corresponding to the number that has been inputted into the program. A total of 14 choices are available to the user which will be discussed in the following section.

3 Functionalities

The user is presented with the following 14 choices on execution:

1. **Fill the registry with random names for testing**
   This option is used to enumerate the program with random attributes of a patient which can be used for testing the program. A further query is done to ask the user for the number of random patients to be added (the one time limit is 500 as to not break the program).

2. **Add new patients**
   This option is used to add patients manually. First the user is asked for the number of patients to be added. Then for each patient, the name, address, age, source and destination is asked.

3. **Allocate all the vacant rooms to the new patients**
   If any vacant rooms are available, all the patients who haven't been assigned a room are allotted one. If a patient was discharged before, this option will not re-allott them a room.

4. **Allocate a vacant room to a specific patient**
   If the user wants to allocate a room to a specific patient, this option can be used. The user will be prompted for the patient's name (which is case insensitive). If the patient is found and a room is available, the patient will be allotted one. This option can also be used to allocate a room to an already discharged patient.

5. **Display the details of all the patients**
   As the name suggests, it displays the details of all the patients available in the records. If the patient has been allotted a room, the room details are also displayed.

6. **Search for the details of a specific patient**
   Searching for the details of a patient can be done through two methods, either by the patient's name or the patient's room number. If the records are found, they are displayed. Otherwise the user is given an error message and is asked to retype the name.

7. **Query the status of a room**
   If this option is selected, the user is asked for the room and floor number of the room being queried. If the room is occupied, the details of the patient occupying it are displayed along with the room. Else, the user is told that the room is vacant.

8. **Display all vacant rooms**
   All the vacant rooms in the quarantine facility are displayed.

9. **Display all occupied rooms**
   All the occupied rooms in the quarantine facility are displayed.

10. **Discharge a specific patient**
    If the user wants to discharge a specific patient, this option can be used. The user will be asked for the patient's name. If the details of the patient are not found in the records, an error message is displayed prompting the user to try again. Else if the details are found, the room being occupied by the patient is made vacant again and the `discharged date` attribute of the patient is automatically updated.

11. **Complete statistics of the quarantine facility**
    On the selection of this choice, the user is presented with the entire statistics of the facility, *i.e.*

    - The total number of occupied rooms
    - The total number of vacant rooms
    - The number of rooms occupied on the ground floor
    - The number of rooms occupied on the first floor
    - The number of rooms occupied on the second floor
    - The total number of patients in the record

12. **Discharge all patients**
    Like the name suggests, all the patients currently occupying the rooms in the facility will be discharged. All other patients will be unaffected.

13. **Sort the patient's list by their respective room numbers**
    The entire available list of the patients will be sorted in an ascending order according to their room numbers and the sorted list will be displayed to the user. All patients with no allotted rooms and patients that have been discharged will be displayed in the beginning of the list.

14. **Exit**
    Used to exit the program.

4 Code Layout

4.1 The `patient` Class

4.1.1 Private Attributes

Variable Name	Data Type	Description
`name`	`string`	Patient's Name
`address`	`string`	Patient's Address
`age`	`int`	Patient's Age
`arrival_date`	`string`	The date and time of entry of the patient
`source`	`string`	The origin city of the patient
`destination`	`string`	The destination city of the patient
`room_serial_no`	`int`	The serial room number of the patient
`room_no`	`int`	The net room number of the patient
`discharged_date`	`string`	The date and time of the patient's discharge
`floor_no`	`int`	The floor number of the patient's room
`id`	`int`	The ID number of the patient
`empty`	`bool`	Is `false` if the details of the patient haven't been given
`room_allotted`	`bool`	Is true if the patient has been allotted a room

4.1.2 Private Methods

Name	Description	Return Type	Parameters
`random_string()`	Returns a random string of a given length	`string`	length `(int)`
`random_number()`	Returns a random number b/w low and high	`int`	low `(int)`, high `(int)`
`convert_date_to_string()`	Returns the current time in `string`	`string`	none

4.1.3 Public Methods

Name	Description	Return Type	Parameters
`display_details()`	Displays all attributes of the patient	`void`	none
`input_details()`	Takes input from the user	`void`	none
`get_room_no()`	Returns room number of the patient	`int`	none
`get_room_serial_no()`	Returns serial room number of the patient	`int`	none
`modify_room_no()`	Modifies the value of room number according to the value passed	`void`	value `(int)`
`get_floor_no()`	Returns the floor number of the patient	`int`	none

`modify_floor_no()`	Modifies the value of floor number	`void`	value`(int)`
`modify_discharged_date`	Modifies discharge date to either "none" or the current time	`void`	a `(bool)`
`get_discharged_date()`	Returns the discharged date of the patient	`string`	none
`get_room_allotted()`	Returns true if the patient has been allotted a room	`bool`	none
`get_age()`	Returns age of the patient	`int`	none
`get_empty()`	Returns false if patient details are empty	`bool`	none
`get_id()`	Returns patient ID	`int`	none
`get_patient_name()`	Returns the name of the patient	`string`	none

4.2 The quarantine class

4.2.1 Private Attributes

Variable Name	Data Type	Description
occ_ground	int	Count of occupied rooms on the ground floor
occ_first	int	Count of occupied rooms on the first floor
occ_second	int	Count of occupied rooms on the second floor
count	int	Total count of patients in the record
allotted_count	int	Total count of occupied rooms
ground	bool[]	Array of the rooms on the ground floor. If a room is vacant, the corresponding array value is false
first	bool[]	Array of the rooms on the first floor
second	bool[]	Array of the rooms on the second floor
p	vector <patient>	Vector of the class patient

4.2.2 Private Methods

Name	Description	Return Type	Parameters
`string_compare()`	Compares two string and returns true if they're equal	`bool`	s1(`string`), s2 (`string`)
`allott_room()`	Allotts a room to a patient according to their age and returns the allotted room number	`int`	age (`int`)
`discharge_room()`	Discharges a patient from a room	`void`	floor_no (`int`), serial_room_no (`int`)
`calc_room_no()`	Calculates the net room number from the serial room number and the floor number	`int`	floor_no (`int`), serial_room_no (`int`)
`check_occupancy()`	Returns true if the passed room is occupied	`bool`	floor_no (`int`), serial_room_no (`int`)
`binary_search()`	Searches for a patient's records using room number, returns true if found	`bool`	room_no (`int`)

4.2.3 Public Methods

Name	Description	Return Type	Parameters
`calc_floor_no()`	Calculates the floor number from the net room number	`int`	room_no `(int)`
`calc_serial_room_no()`	Calculates the serial room number from the net room number	`int`	room_no `(int)`
`add_patients()`	Adds `n` number of patients to the records	`void`	n `(int)`
`fill_patients_test()`	Adds `n` number of random entries to the records for testing	`void`	n `(int)`
`allottment_all()`	Allotts all the vacant rooms to the new patients and returns the total allotted count	`void`	none
`display_all_patients()`	Displays the records of all the patients	`void`	none
`allott_patient()`	Allotts a patient a room and returns the room number allotted	`int`	name `(string)`
`room_details()`	Displays the room details of a specific room	`void`	floor_no `(int)`, room_no `(int)`
`sort_by_room_no()`	Sorts the patients records by their room number	`void`	none

`room_details()`	Displays the room details of a specific room	`void`	floor_no `(int)`, room_no `(int)`
`search_patient()`	Searches the records of a patient by their name or their room number	`bool`	patient_name `(string)` floor_no `(int)`, room_no `(int)`
`display_vacant_rooms()`	Displays the vacant room details, returns false if no rooms are vacant	`bool`	none
`display_occupied_rooms()`	Displays the occupied room details, returns false if no rooms are occupied	`bool`	none
`discharge_patient()`	Discharges a specific patient, returns false if the patient's records are not found	`bool`	name `(string)`
`discharge_all_patients()`	Discharges all patients	`void`	none
`facility_details()`	Displays all current facility details	`void`	none

5 The Code

A working compiled version of the code can be tested at this Repl Repository.

The entire source code is available at this GitHub Repository.

5.1 patient.cpp

```
//PATIENT.CPP
class patient{
   private:
   string name;
   string address;
   int age;
   string arrival_date;
   string source;
   string destination;
   int room_serial_no;
   int room_no;
   string discharged_date;
   int floor_no;
   int id;
   bool empty;
   bool room_allotted;

   string random_string(size_t length){ //generates a random string
       of size length
      auto randchar = []() -> char
      {
      const char charset[] =
      "0123456789"
      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      "abcdefghijklmnopqrstuvwxyz";
      const size_t max_index = (sizeof(charset) - 1);
      return charset[ rand() % max_index ];
      };
      string str(length,0);
      generate_n( str.begin(), length, randchar);
      return str;
```

```
    }

int random_number(int low, int high){ //generates a random
    number bw a given range
   return low + (rand() % high);
}

string convert_date_to_string(){ //returns the current time in
    string format
   auto t = std::time(nullptr);
   auto tm = *std::localtime(&t);
   std::ostringstream oss;
   oss << std::put_time(&tm, "%d-%m-%Y %H-%M-%S");
   auto str = oss.str();
   return str;
}




public:
patient(int i, bool value=false){
   if(value){ //used for generating random entries for testing
   name = random_string(10);
   address = random_string(15);
   age = random_number(12,70);
   arrival_date = convert_date_to_string();
   source = random_string(10);
   destination = random_string(10);
   empty = false;
   room_allotted = false;
   discharged_date = "none";
   room_serial_no = -1;
   room_no = -1;
   floor_no = -1;
}

else{
   name = "none";
   address = "none";
```

```
    age = -1;
    arrival_date = convert_date_to_string();
    source = "none";
    destination = "none";
    room_serial_no = -1;
    room_no = -1;
    discharged_date = "none";
    floor_no = -1;
    empty = true;
    room_allotted = false;
}
    id = i+1;
}

void display_details(){ //displays the details of the patient
    cout<<endl;
    cout<<"Patient Number #"<<id<<endl;
    cout<<"Patient's name : "<<name<<endl;
    cout<<"Address : "<<address<<endl;
    cout<<"Age : "<<age<<endl;
    cout<<"Arrival Date : "<<arrival_date<<endl;
    cout<<"Coming from : "<<source<<endl;
    cout<<"Going to : "<<destination<<endl;
    cout<<"Discharged Date : "<<discharged_date<<endl;
    if(room_allotted){
    cout<<"Allotted Room no. : "<<room_serial_no<<endl;
    cout<<"Floor no. : "<<floor_no<<endl;
}
cout<<endl<<endl;
}

void input_details(){ //takes input from the user for the
    details of the patient
    cout<<"Patient Number #"<<id<<endl;
    cout<<"Please input the Patient details"<<endl;
    cout<<"Name"<<endl;
    getline(cin >> ws, name);
    cout<<"Address"<<endl;
    getline(cin >> ws, address);
    cout<<"Age"<<endl;
```

```
    cin>>age;
    cout<<"Coming from"<<endl;
    getline(cin >> ws, source);
    cout<<"Going to"<<endl;
    getline (cin >> ws, destination);

    empty = false;

}

int get_room_no(){ //returns the room number of the patient
   return room_no;
}

int get_room_serial_no(){ //returns the serial room no of the
    patient
   return room_serial_no;
}

void modify_room_no(int value){ //modifies the room number,
    floor number and room_allotted according to the passed value
if(value == -1){
room_allotted = false;
room_serial_no = -1;
room_no = -1;
}
else{
room_allotted = true;
room_no = value;
if(value <= GROUND_ROOMS){
floor_no = 0;
room_serial_no = room_no;
}
else if(value > GROUND_ROOMS && value <= GROUND_ROOMS +
    FIRST_ROOMS){
floor_no = 1;
room_serial_no = room_no - GROUND_ROOMS;
}
else{
floor_no = 2;
```

```
room_serial_no = room_no - GROUND_ROOMS - FIRST_ROOMS;
}
}
}

int get_floor_no(){ //returns the floor number of the patient
return floor_no;
}

void modify_floor_no(int value){ //modifies the floor number of
    the patient
floor_no = value;
}

void modify_discharged_date(bool a = false){ //modifies the
    discharge date to either none or the current time depending
    on the passed value
if(a)
discharged_date = "none";
else
discharged_date = convert_date_to_string();
}

string get_discharged_date(){ //returns the discharged date of
    the patient
return discharged_date;
}

bool get_room_allotted(){ //returns true if the patient has been
    allotted a room, otherwise false
return room_allotted;
}

int get_age(){ //returns the age of the patient
return age;
}

bool get_empty(){ //returns false if the patient's details
    haven't been filled yet (used as a failsafe)
return empty;
```

```
   }

   int get_id(){ //returns the id of the patient
   return id;
   }

   string get_patient_name(){ //returns the name of the patient
   return name;
   }
};
```

5.2 quarantine.cpp

```
//QUARANTINE.CPP
class quarantine{
private:
int occ_ground;
int occ_first;
int occ_second;
int count;
int allotted_count;
bool ground[GROUND_ROOMS];
bool first[FIRST_ROOMS];
bool second[SECOND_ROOMS];
vector <patient> p;

bool string_compare(string s1, string s2){ //compares two strings
    by converting them both into lowercase and returns true if
    they're equal
if(s1.length()!=s2.length())
return false;

else{
int n = s1.length() - 1;
transform(s1.begin(), s1.end(), s1.begin(), ::tolower);
transform(s2.begin(), s2.end(), s2.begin(), ::tolower);
while(n--){
if(s1[n]!=s2[n])
```

```
return false;
}

return true;
}

}

int allott_room(int age){ //allotts rooms according to the
   patients age and the vacancy of the rooms
if(age >= 60&&occ_ground!=GROUND_ROOMS){
occ_ground++;
for(int i = 0; i < GROUND_ROOMS; i++){
if(ground[i] == false){
ground[i] = true;
return i + 1;
}
}
}
else if(age < 60 && age >= 40 && occ_first<FIRST_ROOMS - 1){
occ_first++;
for(int i = 0; i < FIRST_ROOMS; i++){
if(first[i] == false){
first[i] = true;
return i + GROUND_ROOMS + 1;
}
}
}
else if(age < 40 && occ_second<SECOND_ROOMS - 1){
occ_second++;
for(int i = 0; i < SECOND_ROOMS; i++){
if(second[i] == false){
second[i] = true;
return i + GROUND_ROOMS + FIRST_ROOMS + 1;
}
}
}
return -1;
}
```

```
void discharge_room(int floor_no, int serial_room_no){
   //discharges patients from the rooms
if(floor_no == 0){
occ_ground--;
ground[serial_room_no-1] = false;
}
else if(floor_no == 1){
occ_first--;
first[serial_room_no-1] = false;
}
else{
occ_second--;
second[serial_room_no-1] = false;
}
allotted_count--;
}

int calc_room_no(int floor, int serial_room_no){ //calculates room
   no from serial and floor no
if(floor == 0)
return serial_room_no;
else if(floor == 1)
return serial_room_no + GROUND_ROOMS;
else
return serial_room_no + GROUND_ROOMS + FIRST_ROOMS;
}



bool check_occupancy(int floor, int serial_room_no){ //checks if a
   particular room is occupied or not
if(floor == 0)
return ground[serial_room_no-1];
else if(floor == 1)
return first[serial_room_no-1];
else
return second[serial_room_no-1];
}

bool search(int room_no){ //searches for a room's details
```

```
for(auto& it : p){
if(it.get_room_no() == room_no){
it.display_details();
return true;
}
}

return false;
}

static bool comp_room_no(patient &A, patient &B){ //used for
    sorting the records of the patients by their respective room
    numbers
return (A.get_room_no() < B.get_room_no());
}

public:
quarantine(){
occ_ground = -1;
occ_first = -1;
occ_second = -1;
count = -1;
allotted_count = 0;
for(int i = 0; i < GROUND_ROOMS; i++)
ground[i] = false; //false meaning the room is vacant

for(int i = 0; i < FIRST_ROOMS; i++)
first[i] = false;

for(int i = 0; i < SECOND_ROOMS; i++)
second[i] = false;
}

int calc_floor_no(int room_no){ //calculates the floor number by
    the given room number
if(room_no <= GROUND_ROOMS)
return 0;
else if(room_no > GROUND_ROOMS && room_no <= GROUND_ROOMS +
    FIRST_ROOMS)
return 1;
```

```
else if(room_no > GROUND_ROOMS + FIRST_ROOMS && room_no <=
    GROUND_ROOMS + FIRST_ROOMS + SECOND_ROOMS)
return 2;
else
return -1;
}

int calc_serial_room_no(int room_no){ //calculates the serial
    number by the given room number
if(room_no <= GROUND_ROOMS)
return room_no;
else if(room_no > GROUND_ROOMS && room_no <= GROUND_ROOMS +
    FIRST_ROOMS)
return room_no - GROUND_ROOMS;
else if(room_no > GROUND_ROOMS + FIRST_ROOMS && room_no <=
    GROUND_ROOMS + FIRST_ROOMS + SECOND_ROOMS)
return room_no - GROUND_ROOMS - FIRST_ROOMS;
else
return -1;
}

void add_patients(int n){ //adds new patients to the records
while(n--){
count++;
p.push_back(patient(count));
p[count].input_details();
}
}
void fill_patients_test(int i=30){ //fills the records with random
    entries for testing
while(i--){
count++;
p.push_back(patient(count, true));
}
}

int allottment_all(){ //allotts all the new patients to the vacant
    rooms
if(allotted_count == 500)
return -1;
```

```
else{
for(auto& it : p){
if(allotted_count == 500)
break;
if((!it.get_room_allotted()) &&
    (string_compare(it.get_discharged_date(),"none"))){
int room_no = allott_room(it.get_age());
if(room_no!=-1){
allotted_count++;
it.modify_discharged_date(true);
}
it.modify_room_no(room_no);
}
}
return allotted_count;
}
}

void display_all_patients(){ //displays the data of all patients

for(auto& it : p){
if(!(it.get_empty()))
it.display_details();
}
}


int allott_patient(string patient_name){ //allotts a particular
    patient a vacant room

for(auto& it : p){
if(string_compare(it.get_patient_name(), patient_name)){
if(!it.get_room_allotted()){
int room_no = allott_room(it.get_age());
if(room_no!=-1){
allotted_count++;
it.modify_discharged_date(true);
}
it.modify_room_no(room_no);
```

```
return room_no;
}
}

}
return -1;
}

void room_details(int floor, int room){ //displays the details of
    a room

if(check_occupancy(floor, room)){
cout<<"This room is occupied"<<endl;
cout<<"The patient details are as follows"<<endl;
int room_no = calc_room_no(floor,room);
for(auto& it : p){
if(it.get_room_no() == room_no){
it.display_details();
}
}
}
else{
cout<<"The room is vacant."<<endl;
}
}

void sort_by_room_no(){ //sorts the records of that patients
sort(p.begin(), p.end(), comp_room_no);
}

bool search_patient(string patient_name, int serial_room_no = -1,
    int floor_no = -1){ //searches for the records of a patient
    either by their name
if(serial_room_no == -1){
                                                    //or their
    room number
for(auto& it : p){
if(string_compare(it.get_patient_name(), patient_name)){
it.display_details();
return true;
```

```
}
}
return false;
}
else{
int room_no = calc_room_no(floor_no, serial_room_no);
return search(room_no);
}
}

bool display_vacant_rooms(){ //displays all vacant rooms
int room_no = 500;
bool flag = false;
while(room_no){
if(!check_occupancy(calc_floor_no(room_no),
    calc_serial_room_no(room_no))){
cout<<"Room no: "<<calc_serial_room_no(room_no)<<endl;
cout<<"Floor no: "<<calc_floor_no(room_no)<<endl<<endl;
flag = true;
}
room_no--;
}

return flag;
}


bool display_occupied_rooms(){ //displays all occupied rooms
int room_no = 500;
bool flag = false;
while(room_no){
if(check_occupancy(calc_floor_no(room_no),
    calc_serial_room_no(room_no))){
cout<<"Room no: "<<calc_serial_room_no(room_no)<<endl;
cout<<"Floor no: "<<calc_floor_no(room_no)<<endl<<endl;
flag = true;
}
room_no--;
}
```

```
return flag;
}

bool discharge_specific_patient(string patient_name){ //discharges
    a specific patient
for(auto& it : p){
if(string_compare(it.get_patient_name(), patient_name)){
if(it.get_room_allotted()){
discharge_room(it.get_floor_no(), it.get_room_serial_no());
cout<<"This patient is being discharged"<<endl;
it.modify_room_no(-1);
it.modify_discharged_date();
it.display_details();
}

else{
cout<<"The patient hasn't been allotted a room"<<endl;
}
return true;
}
}
return false;
}


void discharge_all_patients(){ //discharges all patients available
    in the records
for(auto& it : p){
if(it.get_room_allotted()){
discharge_room(it.get_floor_no(), it.get_room_serial_no());
cout<<"This patient is being discharged"<<endl;
it.modify_room_no(-1);
it.modify_discharged_date();
it.display_details();
}
}
}
void facility_details(){ //displays all the details of the facility
cout<<"Total rooms occupied: "<<allotted_count<<endl;
cout<<"Total rooms vacant: "<<500-allotted_count<<endl;
```

```
cout<<"Rooms occupied on ground floor: "<<occ_ground+1<<endl;
cout<<"Rooms occupied on first floor: "<<occ_first+1<<endl;
cout<<"Rooms occupied on second floor: "<<occ_second+1<<endl;
cout<<"Total patients in the record: "<<count+1<<endl;
}
};
```

5.3 main.cpp

```
//MAIN.CPP
const int TOTAL_ROOMS = 500;
const int GROUND_ROOMS = 200;
const int FIRST_ROOMS = 150;
const int SECOND_ROOMS = 150; //The values of these constants can
    be changed as it seems fit

using namespace std;

#include<iostream>
#include<string>
#include<bits/stdc++.h>
#include<vector>
#include<ctype.h>
#include<ctime>
#include<random>
#include <iomanip>
#include <sstream>
#include<algorithm>
#include"patient.cpp" //contains the patient class
#include"quarantine.cpp" //contains the quarantine class

int main(){
srand((unsigned) time(0)); //seeding the rand function
quarantine facility;
char choice ='y';
int option = -1, n = -1;
bool flag = false, flag2 = true; //these variables are used in
    getting the input from the user
```

```
string s;

do{

if(flag2)
cout<<endl<<endl<<"*****************Welcome to the quarantine
    facility management system of NIT
    SILCHAR***************"<<endl<<endl;
cout<<"What do you want to do?"<<endl;
cout<<"Please input the number corresponding to your
    choice:"<<endl;
cout<<"1. Fill the registry with random names for testing"<<endl;
cout<<"2. Add new patients"<<endl;
cout<<"3. Allocate all the vacant rooms to the new patients"<<endl;
cout<<"4. Allocate a vacant room to a specific patient"<<endl;
cout<<"5. Display the details of all the patients"<<endl;
cout<<"6. Search for the details of a specific patient"<<endl;
cout<<"7. Query the status of a room"<<endl;
cout<<"8. Display all vacant rooms"<<endl;
cout<<"9. Display all occupied rooms"<<endl;
cout<<"10. Discharge a specific patient"<<endl;
cout<<"11. Complete statistics of the quarantine facility"<<endl;
cout<<"12. Discharge all patients"<<endl;
cout<<"13. Sort the patient's list by their respective room
    numbers"<<endl;
cout<<"14. Exit"<<endl;
cin>>option;
switch(option){
case 1:
do{
flag = false;
cout<<endl;
cout<<"The registry will be filled with random data for your
    convenience"<<endl;
cout<<"Please input the number of patients you want to be added
    (should not be greater than 500)"<<endl;
cin>>n;
if(n > 500 || n <= 0){ //if entered value is out of bounds, user
    is asked to try again
flag = true;
```

```
cout<<"Incorrect input, please try again."<<endl;
}
else
facility.fill_patients_test(n);
}while(flag);
break;

case 2:
do{
flag = false;
cout<<endl;
cout<<"Please input the number of patients you want to add (should
    not be greater than 500)"<<endl;
cin>>n;
if(n > 500 || n <= 0){
flag = true;
cout<<"Incorrect input, please try again."<<endl;
}
else
facility.add_patients(n);
}while(flag);
break;

case 3:
cout<<endl;
cout<<"All the vacant rooms will be allotted to the new
    patients"<<endl;
n = facility.allottment_all();
if(n == -1)
cout<<"There are no vacant rooms, so no new rooms were
    allotted."<<endl;
else{
cout<<"The total number of allotted rooms is now "<<n<<endl;
}
break;

case 4:
cout<<endl;
cout<<"Please enter the name of the patient (case
    insensitive)"<<endl;
```

```
getline(cin >> ws, s);
n = facility.allott_patient(s);
if(n!=-1){
cout<<"The patient's record was found and he was allotted in
    "<<endl;
cout<<"Room no: "<<facility.calc_serial_room_no(n)<<endl;
cout<<"Floor no: "<<facility.calc_floor_no(n)<<endl;
}
else
cout<<"Either the patient is already allotted in a room or the
    patient's record do not exist"<<endl;
break;

case 5:
cout<<endl;
cout<<"All available patient records will be displayed."<<endl;
facility.display_all_patients();
break;

case 6:
cout<<endl;
n = -1;
cout<<"Do you want to search through:"<<endl;
cout<<"1. Patient's name"<<endl;
cout<<"2. Patient's Room no and Floor no"<<endl;
cin>>n;
if(n==1){
cout<<"Please enter the patient's name (case insensitive)"<<endl;
getline(cin >> ws, s);
flag = facility.search_patient(s);
}
else{
int serial_room_no, floor_no;
cout<<"Please enter the patient's serial room no"<<endl;
cin>>serial_room_no;
cout<<"Please enter the patient's floor no (0 for Ground, 1 for
    First, 2 for Second)"<<endl;
cin>>floor_no;
flag = facility.search_patient("none", serial_room_no,floor_no);
}
```

```
if(!flag)
cout<<"Patient's record was not found"<<endl;
break;

case 7:
do{
flag = false;
int serial_room_no, floor_no;
cout<<"Please enter the room no"<<endl;
cin>>serial_room_no;
cout<<"Please enter the floor no (0 for Ground, 1 for First, 2 for
    Second)"<<endl;
cin>>floor_no;
if(floor_no > 2 || floor_no < 0 || serial_room_no > 200 ||
    serial_room_no <= 0)
flag = true;
if((floor_no == 1 || floor_no == 2) && (serial_room_no > 150))
flag = true;
if(!flag){
facility.room_details(floor_no, serial_room_no);
}
else{
cout<<"Incorrect details, please try again"<<endl;
}
}while(flag);

break;


case 8:
cout<<"Displaying vacant rooms"<<endl;
flag = facility.display_vacant_rooms();
if(!flag)
cout<<"No vacant rooms were found"<<endl;

break;

case 9:
cout<<"Displaying occupied rooms"<<endl;
flag = facility.display_occupied_rooms();
```

```
if(!flag)
cout<<"No occupied rooms were found"<<endl;
break;

case 10:
cout<<endl;
cout<<"Please enter the name of the patient you want to discharge
    (case insensitive)"<<endl;
getline(cin >> ws, s);
flag = facility.discharge_specific_patient(s);
if(!flag){
cout<<"The patient's record were not found"<<endl;
}
break;

case 11:
facility.facility_details();
break;

case 12:
facility.discharge_all_patients();
break;

case 13:
facility.sort_by_room_no();
cout<<"The list has been sorted"<<endl;
facility.display_all_patients();
break;

case 14:
cout<<"Are you sure you want to exit? (y/n)"<<endl;
cin>>choice;
if(choice=='y'||choice=='Y')
return 0;

break;


default:
cout<<"Incorrect input, please try again"<<endl;
```

```
}
flag2 = false;
cout<<"Continue? (y/n)"<<endl;
cin>>choice;
}while(choice == 'y' || choice == 'Y');
return 0;

}
```

6 Advantages

- Since the program was written keeping *Object Oriented Programming* practices in mind, it is substantially easy for anyone to add, remove and modify it's features.
- The `patient` and `quarantine` class have abstracted all the irrelevant details from the user, and encapsulated the attributes and methods into two classes.
- The attributes of the patients can't be directly edited and accessed by the user. Therefore, the records can't be corrupted or edited mistakenly by anyone.
- The capacity of the floors were declared as a constant global variable, meaning this program can be quickly modified for the use in a building with different specifications.
- This program is extremely flexible and can be quickly adapted for other facilities. More floors can be added or removed, the attribute specifications can also be changed since `patient` is an independent class, and similarly new methods can be introduced in the `quarantine` class.

7 Shortcomings

In the making and after the completion of this program, several shortcomings in the code were noticed which can be improved upon

- Since the data inputted in the program exists as long as the program is executing, an actual backend should be implemented in tandem with this application.
- File handling can be introduced to store and save the data for long term use.
- The interface with the user can be modified and upgraded since a serial number based menu isn't feasible in the long term.
- Exceptions weren't used in this program, hence a character input where an integer is expected can break the program.
- Due to the fact that the program will most probably not be run straight for 14 days, there's no existing feature that will automatically discharge the patients after a specific period of time.

8 Conclusion

Hence, a program to handle the quarantine facility in NIT Silchar was successfully completed. In the process of coding this program, we further strengthened our grasp on the topics of *Data Structures*, *Algorithms* and *OOPS*. It was an enjoyable and learning experience for all the students involved in this project. Also, several improvements can be done to this program to make it viable for practical use.