UNIVERSIDAD PANAMERICANA

UNIVERSIDAD PANAMERICANA
Campus México

Profesor: Francisco Javier Aguilar Juárez

Materia: Programación y Estructura de Datos (COM104)

Fecha de entrega: 15 de noviembre de 2022

Ciclo: 1228

Nombre del proyecto: SchedulEmUP

Miembros del equipo		
ID	**Nombre**	**Carrera**
0245501	Gabriela Shaooli Cassab	LIDCI
0241823	Enrique Ulises Báez Gómez Tagle	LIDCI
0212508	Carlos Isunza Frank	LIDCI
0212570	Santiago Valdés Uriarte	LIDCI

Análisis / Analysis

Descripción detallada del problema a resolver.

Muchas veces, en los hospitales se tiene un control deficiente de la información en general, y esto resulta en confusiones, en el mejor de los casos, pero en el peor de los casos puede provocar pérdida de información, resultando en errores que pueden costar vidas. Nosotros buscamos ponerle fin a este fenómeno que frena la productividad.

Decidimos crear una aplicación en la cual se lleve el registro de todos los doctores disponibles en el hospital, para luego poder asignarle a cada doctor sus pacientes de acuerdo al orden de llegada. Con lo anterior queremos lograr que sea fácil consultar y manejar la información, pero que al mismo tiempo los doctores puedan dar de alta pacientes y filtrar la lista general de acuerdo a la urgencia.

Detailed description of the problem to be solved.

Many times in hospitals there's poor management of information, and this may result in confusion, in the best of cases, but in the worst of cases, it can cause loss of information resulting in errors that can cost lives. We seek to put an end to this phenomenon that slows down productivity and endangers lives.

We decided to create an application in which the record of all the doctors available in the hospital is kept so that we can then assign patients according to their arrival order and with their reason for admission into the hospital. Having these, we want to make it easier to consult and manage the information, but at the same time help doctors to prioritize more time-sensitive cases by filtering the patient list by urgency, to ensure patients' health and to have a more efficient time management.

Diseño / Design

Descripción de las estructuras de datos que se emplearán y la explicación de cómo estas estructuras resuelven el problema.

En este proyecto utilizamos dos estructuras de datos: listas y colas. Usamos éstas ya que consideramos que la presente adaptación era lo más cercano a cómo funcionan estos sistemas en el mundo real. En primera instancia, elegimos una ‘lista’ de pacientes que se comporta como cola (pues solo inserta al final) pero que tiene la capacidad eliminar datos desde cualquier posición. Por otro lado, diseñamos una lista de doctores donde cargamos a cada uno de los médicos disponibles. Por último, cada doctor tiene su ‘cola’ en donde se registran los pacientes de acuerdo al médico que visitarán.

Al ser tratados los pacientes por su doctor, se puede dar de alta al paciente tanto de la cola del doctor, como de la lista, a través de su ID. En la lista enlazada, realizamos el procedimiento normal para eliminar un nodo de acuerdo a un campo en específico. Sin embargo, en la fila esto es posible, aunque el esquema de esta estructura es PEPS (Primera Entrada Primera Salida), ya que la lógica que usamos para extraer a un paciente específico fue desencolar a cada uno de los pacientes, comparar su ID con el que buscamos, en caso afirmativo, eliminarlo, y en caso negativo, regresarlo a la cola, para así no alterar el orden con el que comenzamos. Por último, otra de las opciones que ofrece nuestro programa es filtrar y mostrar a todos los pacientes del hospital que estén registrados como urgentes.

A description of the data structures that will be used and how they will resolve the problem.

In this project, we use two types of data structures: lists and queues. We chose these two structures specifically since they are the closest to what real hospitals use when trying to solve a system of patients arriving at the hospital. We begin by making a 'list' that acts as a queue (only inserting at the tail) but having the capacity to erase anywhere, then we have a list of doctors available each having a 'queue' of patients who will carry out their consultations with them.

When patients are treated by their doctor, the patient can be discharged both from the doctor's queue and from the list, through their ID. In the linked list, we perform the normal procedure to remove a node according to a specific field. However, in the queue this is possible, although the scheme of this structure is FIFO (First In First Out), since the logic we used to extract a specific patient was to dequeue each one of the patients, compare their ID with the one we are looking for, if so, delete it, and if not, return it to the queue, so as not to alter the order with which we started.

Finally, another of the options offered by our program is to filter and show all hospital patients who are registered as urgent.

Implementación / Implementation

Código completo / Complete code

Archivo / File	Código / Code

main.cpp	
	```
	#include <iostream>
	#include "main_menu.h"

	using namespace std;

	int main()
	{
		main_menu();
		return 0;
	}
	```

main_menu.h	
	```
	#pragma once
	#include "add_patient.h"
	#include "list.h"
	#include "load_doctors.h"
	#include "doctor_queues.h"
	#include "list_4_patients.h"
	#include "search_delete_queues.h"


	enum main_menu
	{
		EXIT = 0,
		Add_Patient,
		see_doctors,
		see_doctor_queues,
		discharge_patient,
		see_all_patients,
		see_urgent,
	};

	inline void main_menu()
	{
		queue queue_john, queue_mary, queue_peter;
		list doctors;
		list_4_patients complete_patients;
		load_doctors(doctors);
		patient new_patient;
		int choice;
		do
		{
			cout << endl << endl << "MAIN MENU" <<
	```

```
endl;
                cout << "1. Add patient" << endl;
                cout << "2. See doctors" << endl;
                cout << "3. See doctor's queue" << endl;
                cout << "4. Discharge Patient by ID" << endl;
                cout << "5. See All Patients" << endl;
                cout << "6. See Urgent Patients" << endl;
                cout << "0) EXIT" << endl;

                cout << "Choice: ";
                cin >> choice;

                switch (choice)
                {
                case Add_Patient:
                        new_patient = add_patient();
                        if (new_patient.doctor_2_visit == 1)
queue_john.enqueue(new_patient);
                        else if (new_patient.doctor_2_visit ==
2) queue_mary.enqueue(new_patient);
                        else if (new_patient.doctor_2_visit ==
3) queue_peter.enqueue(new_patient);
                        else
                        {
                                cout << "Invalid doctor ID" <<
endl;
                                new_patient = add_patient();
                        }

complete_patients.insert(new_patient);
                        break;

                case see_doctors:
                        doctors.show_full();
                        break;

                case see_doctor_queues:
                        doctor_queues(queue_john,
queue_mary, queue_peter);
                        break;

                case discharge_patient:
                        int id;
                        cout << "Enter ID: ";
                        cin >> id;
                        complete_patients.delete_id(id);
                        search_delete_queues(queue_john,
queue_mary, queue_peter, id);
                        break;
                case see_all_patients:
                        complete_patients.show_full();
                        break;

                case see_urgent:
                        complete_patients.show_urgent();
                        break;

                case EXIT:
                        cout << "EXITING PROGRAM" <<
endl;
                        break;
                default:
                        cout << "Invalid choice" << endl;
                        break;
                }
        }
        while (choice != 0);
}
```

doctor.h

```
#pragma once
#include <string>
using namespace std;

class doctor
{
public:
        doctor();
        doctor(string, int, int, string);

        string name{};
        int age{};
        int dr_id{};
        string specialty{};

        void details() const;
};
```

doctor.cpp

```
#include "doctor.h"

#include <iostream>
using namespace std;

doctor::doctor() = default;

doctor::doctor(string name, const int age, const int dr_id,
string specialty)
{
```

	```
			this->name = move(name);
			this->age = age;
			this->dr_id = dr_id;
			this->specialty = move(specialty);
	}

	void doctor::details() const
	{
			cout << "Name: " << name << "\tAge: " << age << "\tID: " << dr_id << "\tSpecialty: " << specialty << endl;
	}
	```

patient.h	
	```
	#pragma once
	#include <string>
	#include <iostream>
	using namespace std;

	class patient
	{
	public:
			patient();
			patient(int, string, int, int, string, int);

			int id{};
			string name{};
			int age{};
			int doctor_2_visit{};
			string cause;
			int urgency{};

			int get_id() const;

			void details() const
			{
					cout << "PATIENT ID: " << id << "\t" << "NAME: " << name << "\t" << "AGE: " << age << "\t" <<
							"DOCTOR TO VISIT: " << doctor_2_visit << "\t" << "CAUSE: " << cause << "\t" << "URGENCY: " << urgency <<
							endl;
			}
	};
	```

patient.cpp	
	```
	#include "patient.h"

	#include <iostream>
	using namespace std;

	patient::patient() = default;
	```

	```
	patient::patient(const int id, string name, const int age, const int doctor_2_visit, string cause, const int urgency)
	{
		this->id = id;
		this->name = move(name);
		this->age = age;
		this->doctor_2_visit = doctor_2_visit;
		this->cause = move(cause);
		this->urgency = urgency;
	}

	int patient::get_id() const
	{
		return id;
	}
	```

list.h	
	```
	#pragma once
	#include "doctor.h"

	class list
	{
	public:
		list();
		~list();

		void insert(const doctor&);
		void show_full();
		bool is_empty() const;

	private:
		struct node
		{
			doctor data;
			node* next{};
		};

		node *head_, *tail_, *temp_;
	};
	```

list.cpp	
	```
	#include "list.h"
	#include <iostream>
	using namespace std;

	list::list() { head_ = tail_ = temp_ = nullptr; }

	list::~list()
	{
		while (head_)
	```

	```
	&#123;
			temp_ = head_;
			head_ = head_->next;
			delete temp_;
		&#125;
	&#125;

	void list::insert(const doctor& doc)
	&#123;
		temp_ = new node;
		temp_->data = doc;

		if (head_ == nullptr)
			head_ = tail_ = temp_;

		else
		&#123;
			tail_->next = temp_;
			tail_ = temp_;
			tail_->next = nullptr;
		&#125;
	&#125;

	void list::show_full()
	&#123;
		if (is_empty())
		{
			cout << "List is empty" << endl;
			return;
		&#125;
		temp_ = head_;
		while (temp_->next != nullptr)
		&#123;
			temp_->data.details();
			temp_ = temp_->next;
		&#125;
		temp_->data.details();
	}

	bool list::is_empty() const
	{
		return head_ == nullptr;
	&#125;
	```

queue.h	
	```
	#pragma once
	#include "patient.h"

	struct node
	{
		patient data;
	```

```
	node* next{};
};

class queue
{
public:
	queue();
	~queue();
	void enqueue(patient);
	patient dequeue();

	void show() const;
	bool is_empty() const;
	int size() const;
	void delete_id(int);
	void clear();

private:
	node* head_;
	node* tail_;
};
```

queue.cpp

```
#include "queue.h"

#include <utility>

using namespace std;

queue::queue()
{
	head_ = nullptr;
	tail_ = nullptr;
}

queue::~queue()
{
	const node* current = head_;
	while (current != nullptr)
	{
		const node* next = current->next;
		delete current;
		current = next;
	}
}

void queue::enqueue(patient new_)
{
	const auto new_node = new node;
	new_node->data = move(new_);
	new_node->next = nullptr;
```

```
    if (head_ == nullptr)
    {
        head_ = new_node;
        tail_ = new_node;
    }
    else
    {
        tail_->next = new_node;
        tail_ = new_node;
    }
}

patient queue::dequeue()
{
    if (head_ == nullptr) return patient{};

    const auto old_head = head_;
    const auto old_data = old_head->data;
    head_ = old_head->next;
    delete old_head;

    return old_data;
}


void queue::show() const
{
    if (is_empty())
    {
        cout << "Queue is empty" << endl;
        return;
    }

    const node* current = head_;
    while (current->next != nullptr)
    {
        current->data.details();
        current = current->next;
    }
    current->data.details();
}

bool queue::is_empty() const
{
    return head_ == nullptr;
}

int queue::size() const
{
    int size = 0;
    const node* current = head_;
    while (current != nullptr)
    {
        size++;
        current = current->next;
    }
    return size;
}

void queue::delete_id(const int id)
{
    int count = 0;
    while (count < size())
    {
        patient temp = dequeue();
        count++;
        if (temp.get_id() != id)
        {
            enqueue(temp);
        }
    }
}

void queue::clear()
{
    const node* current = head_;
    while (current != nullptr)
    {
        const node* next = current->next;
        delete current;
        current = next;
    }
    head_ = nullptr;
    tail_ = nullptr;
}
```

list_4_patients.h

```
#pragma once
#include "patient.h"

class list_4_patients
{
public:
    list_4_patients();
    ~list_4_patients();

    void insert(const patient&);
    void show_full() const;
    bool is_empty() const;
    void delete_id(int);
    patient get_head() const;
    void pop_front();
    void show_urgent() const;

private:
    struct node
    {
        patient data;
        node* next{};
    };

    node *head_, *tail_, *temp_;
};
```

list_4_patients.cpp

```
#include "list_4_patients.h"
#include <iostream>
using namespace std;


list_4_patients::list_4_patients() { head_ = tail_ = temp_ =
nullptr; }

list_4_patients::~list_4_patients()
{
    while (head_)
    {
        temp_ = head_;
        head_ = head_->next;
        delete temp_;
    }
}

void list_4_patients::insert(const patient& doc)
{
    temp_ = new node;
    temp_->data = doc;

    if (head_ == nullptr)
        head_ = tail_ = temp_;

    else
    {
        tail_->next = temp_;
        tail_ = temp_;
        tail_->next = nullptr;
    }
}
```

```
void list_4_patients::show_full() const
{
    if (is_empty())
    {
        cout << "List is empty" << endl;
        return;
    }
    const node* temp = head_;
    while (temp != nullptr)
    {
        temp->data.details();
        temp = temp->next;
    }
}

bool list_4_patients::is_empty() const
{
    return head_ == nullptr;
}


void list_4_patients::delete_id(const int id)
{
    if (is_empty())
    {
        cout << "List is empty" << endl;
        return;
    }
    node* current = head_;
    node* previous = nullptr;
    while (current != nullptr)
    {
        if (current->data.id == id)
        {
            if (current == head_)
            {
                head_ = head_->next;
                delete current;
                return;
            }
            if (current == tail_)
            {
                tail_ = previous;
                tail_->next = nullptr;
                delete current;
                return;
            }
            previous->next = current->next;
            delete current;
            return;
        }
        previous = current;
        current = current->next;
    }

    cout << "ID not found" << endl;
}

patient list_4_patients::get_head() const
{
    return head_->data;
}

void list_4_patients::pop_front()
{
    if (is_empty())
    {
        cout << "List is empty" << endl;
        return;
    }
    const node* temp = head_;
    if (head_ == tail_)
        head_ = tail_ = nullptr;
    else
        head_ = head_->next;
    delete temp;
}

void list_4_patients::show_urgent() const
{
    if (is_empty())
    {
        cout << "List is empty" << endl;
        return;
    }
    cout << "Urgent Patients" << endl;
    const node* temp = head_;
    while (temp != nullptr)
    {
        if (temp->data.urgency == 1)
            temp->data.details();
        temp = temp->next;
    }
}
```

add_patient.h

```
#pragma once
#include "patient.h"

inline patient add_patient()
{
	int id;
	string name;
	int age;
	int doctor_2_visit;
	string cause;
	int urgency;

	cout << "ID: ";
	cin >> id;
	cout << "Name: ";
	cin >> name;
	cout << "Age: ";
	cin >> age;
	cout << "Doctor to visit: " << endl;
	cout << "1. Dr. John" << endl;
	cout << "2. Dr. Mary" << endl;
	cout << "3. Dr. Peter" << endl;
	cout << "Select doctor ID: ";
	cin >> doctor_2_visit;
	cout << "Cause: ";
	cin >> cause;
	cout << "Urgency (1 for VERY URGENT - 2 for
normal): ";
	cin >> urgency;

	patient new_patient(id, name, age, doctor_2_visit,
cause, urgency);
	return new_patient;
}
```

doctor_queues.h

```
#pragma once
#include <iostream>

#include "queue.h"
using namespace std;

enum doctor_queues
{
	back = 0,
	see_john,
	see_mary,
	see_peter
};

inline void doctor_queues(const queue& queue_john, const
queue& queue_mary, const queue& queue_peter)
{
	cout << "Available queues:" << endl;
```

```
        cout << "1. John" << endl;
        cout << "2. Mary" << endl;
        cout << "3. Peter" << endl;
        cout << "0. EXIT" << endl;

        int choice;
        cout << "Choice: ";
        cin >> choice;

        switch (choice)
        {
        case see_john:
                queue_john.show();
                break;

        case see_mary:
                queue_mary.show();
                break;

        case see_peter:
                queue_peter.show();
                break;

        case back:
                cout << "Exiting..." << endl;
                break;

        default:
                cout << "Invalid choice" << endl;
                break;
        }
}
```

load_doctors.h

```
#pragma once
inline void load_doctors(list& doctors)
{
        const auto doctor1 = doctor("John", 30, 1,
"Cardiology");
        const auto doctor2 = doctor("Mary", 40, 2,
"Neurology");
        const auto doctor3 = doctor("Peter", 50, 3,
"Dermatology");
        doctors.insert(doctor1);
        doctors.insert(doctor2);
        doctors.insert(doctor3);
        cout << "DOCTORS LOADED" << endl;
}
```

search_delete_queues.h

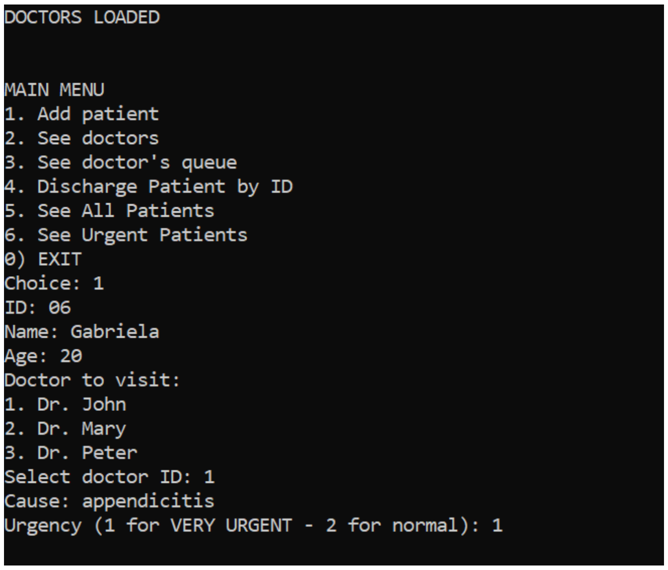
```
#pragma once
#include <iostream>
```

	#include "queue.h"
	
	using namespace std;
	
	inline void search_delete_queues(queue& queue_john, queue& queue_mary, queue& queue_peter, const int id)
	{
	    queue_john.delete_id(id);
	    queue_mary.delete_id(id);
	    queue_peter.delete_id(id);
	}

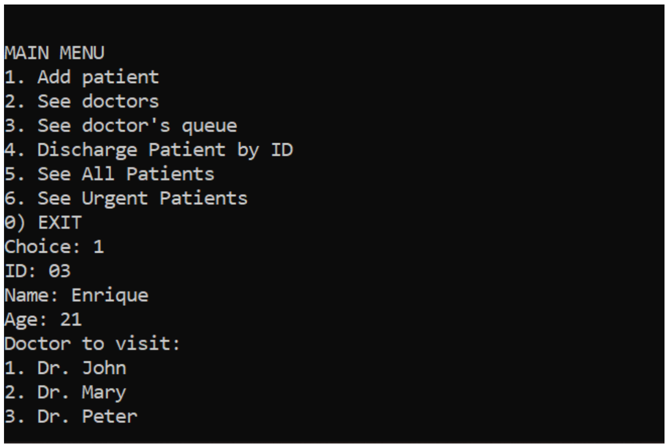
Prueba / Test

Imagen de algunas pantallas que ejemplifican la ejecución de la aplicación.
Some screenshots that exemplify the execution of the application.

Opción 1: Agregar paciente / Option 1: Add patient
Agregamos a los pacientes Enrique y Gabriela con ID 3 y 6 respectivamente.
We added patients ‘Enrique’ and ‘Gabriela’ with IDs 3 and 6 respectively.

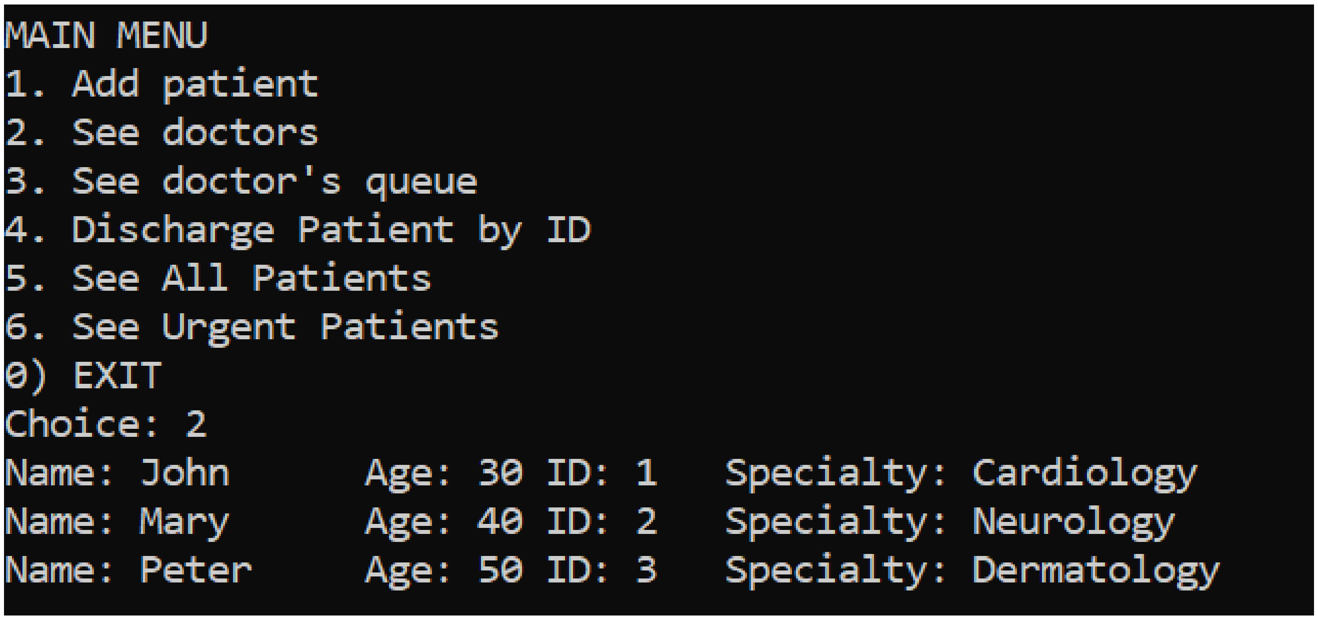


```
DOCTORS LOADED


MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 1
ID: 06
Name: Gabriela
Age: 20
Doctor to visit:
1. Dr. John
2. Dr. Mary
3. Dr. Peter
Select doctor ID: 1
Cause: appendicitis
Urgency (1 for VERY URGENT - 2 for normal): 1
```



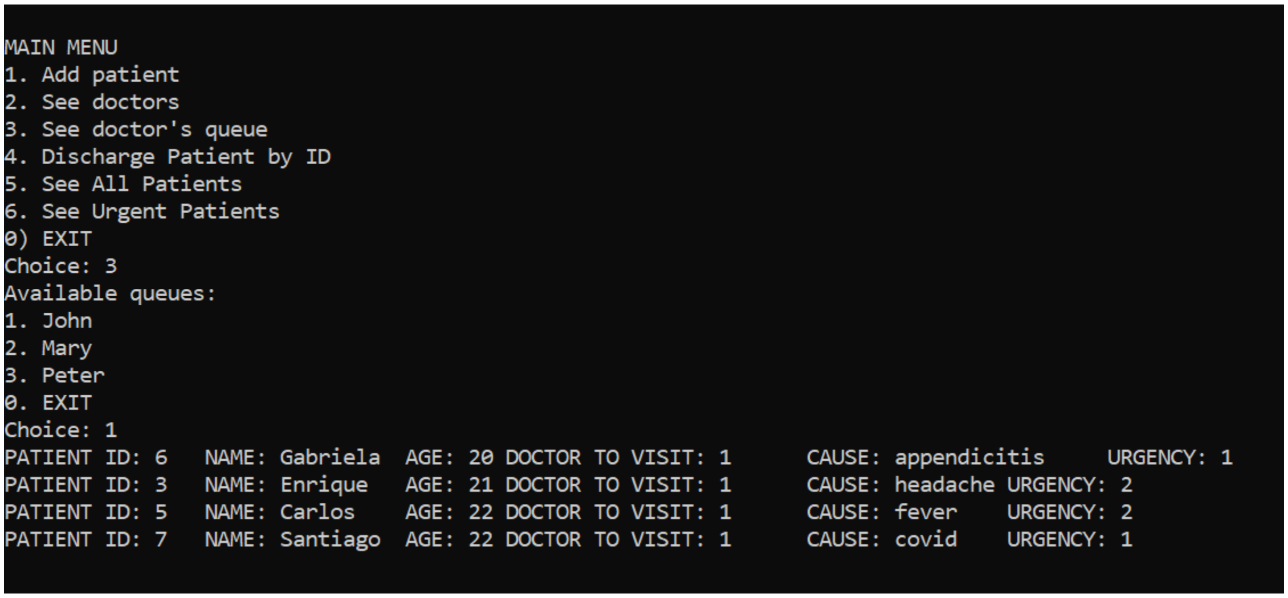
```
MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 1
ID: 03
Name: Enrique
Age: 21
Doctor to visit:
1. Dr. John
2. Dr. Mary
3. Dr. Peter
```

Opción 2: Ver doctores / Option 2: See doctors
Podemos observar a los 3 doctores del hospital, su edad, ID, y especialidad.
We can observe the 3 doctors of the hospital, their age, ID and specialty.



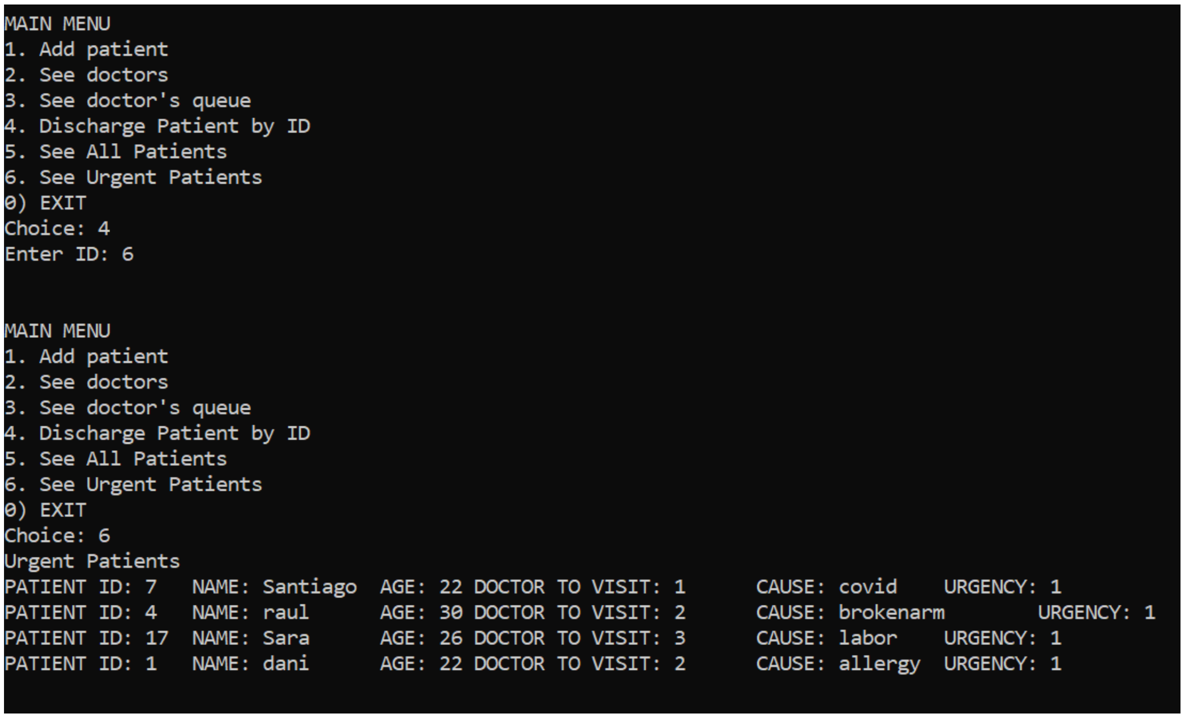
```
MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 2
Name: John      Age: 30 ID: 1   Specialty: Cardiology
Name: Mary      Age: 40 ID: 2   Specialty: Neurology
Name: Peter     Age: 50 ID: 3   Specialty: Dermatology
```

Opción 3: Ver cola de doctor / Option 3: See doctor's queue
Elegimos ver la cola de pacientes que visitan al doctor John (ID 1).
We chose to see Dr. John's patient queue.



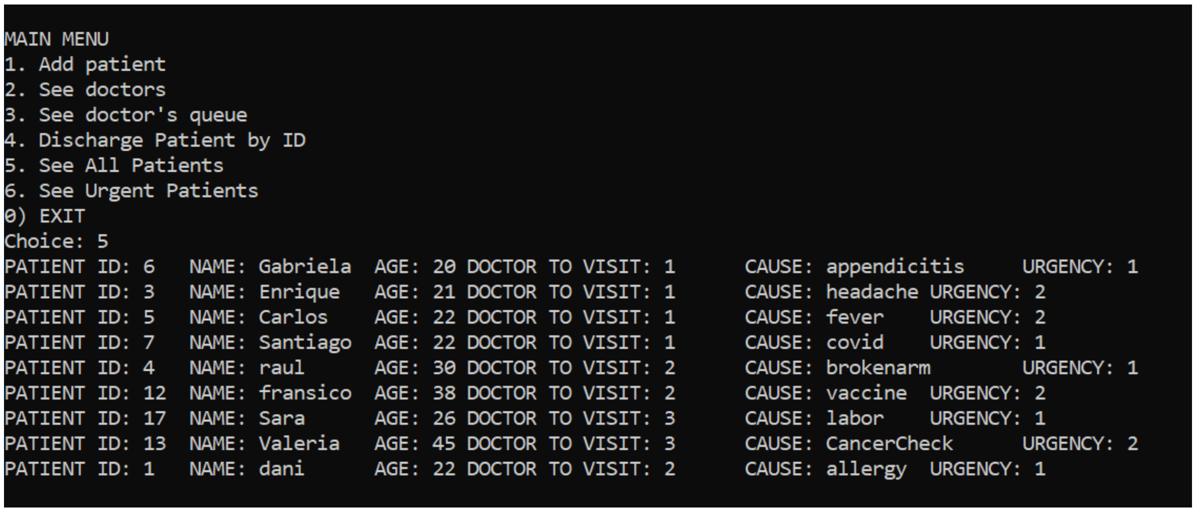
```
MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 3
Available queues:
1. John
2. Mary
3. Peter
0. EXIT
Choice: 1
PATIENT ID: 6   NAME: Gabriela  AGE: 20 DOCTOR TO VISIT: 1      CAUSE: appendicitis     URGENCY: 1
PATIENT ID: 3   NAME: Enrique   AGE: 21 DOCTOR TO VISIT: 1      CAUSE: headache URGENCY: 2
PATIENT ID: 5   NAME: Carlos    AGE: 22 DOCTOR TO VISIT: 1      CAUSE: fever    URGENCY: 2
PATIENT ID: 7   NAME: Santiago  AGE: 22 DOCTOR TO VISIT: 1      CAUSE: covid    URGENCY: 1
```

Opción 4: Eliminar paciente por ID / Option 4: Discharge Patient by ID
Dimos de alta a la paciente con ID 6 (Gabriela). Ya no se despliega después en la lista de pacientes.
We discharged the patient with ID 6 (Gabriela). It is no longer displayed later in the patient's list.



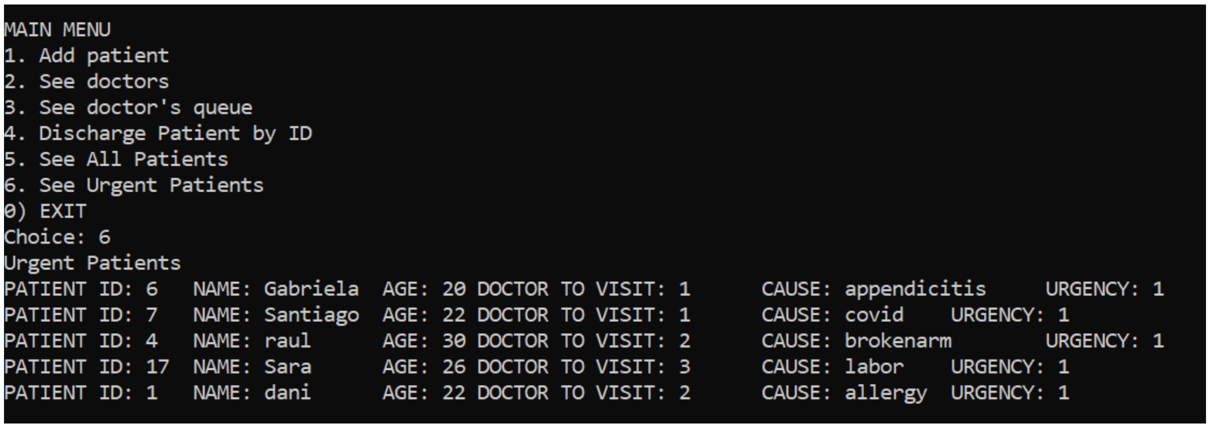
```
MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 4
Enter ID: 6


MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 6
Urgent Patients
PATIENT ID: 7   NAME: Santiago  AGE: 22 DOCTOR TO VISIT: 1      CAUSE: covid    URGENCY: 1
PATIENT ID: 4   NAME: raul      AGE: 30 DOCTOR TO VISIT: 2      CAUSE: brokenarm        URGENCY: 1
PATIENT ID: 17  NAME: Sara      AGE: 26 DOCTOR TO VISIT: 3      CAUSE: labor    URGENCY: 1
PATIENT ID: 1   NAME: dani      AGE: 22 DOCTOR TO VISIT: 2      CAUSE: allergy  URGENCY: 1
```

Opción 5: Ver todos los pacientes / Option 5: See all patients
Se despliega una lista que contiene a todos los pacientes.
A list containing all patients is displayed.



```
MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 5
PATIENT ID: 6   NAME: Gabriela  AGE: 20 DOCTOR TO VISIT: 1      CAUSE: appendicitis     URGENCY: 1
PATIENT ID: 3   NAME: Enrique   AGE: 21 DOCTOR TO VISIT: 1      CAUSE: headache URGENCY: 2
PATIENT ID: 5   NAME: Carlos    AGE: 22 DOCTOR TO VISIT: 1      CAUSE: fever    URGENCY: 2
PATIENT ID: 7   NAME: Santiago  AGE: 22 DOCTOR TO VISIT: 1      CAUSE: covid    URGENCY: 1
PATIENT ID: 4   NAME: raul      AGE: 30 DOCTOR TO VISIT: 2      CAUSE: brokenarm        URGENCY: 1
PATIENT ID: 12  NAME: fransico  AGE: 38 DOCTOR TO VISIT: 2      CAUSE: vaccine  URGENCY: 2
PATIENT ID: 17  NAME: Sara      AGE: 26 DOCTOR TO VISIT: 3      CAUSE: labor    URGENCY: 1
PATIENT ID: 13  NAME: Valeria   AGE: 45 DOCTOR TO VISIT: 3      CAUSE: CancerCheck      URGENCY: 2
PATIENT ID: 1   NAME: dani      AGE: 22 DOCTOR TO VISIT: 2      CAUSE: allergy  URGENCY: 1
```

Opción 6: Ver pacientes urgentes / Option 6: See urgent patients
Se despliega una lista que contiene a los pacientes urgentes.
A list containing urgent patients is displayed.



```
MAIN MENU
1. Add patient
2. See doctors
3. See doctor's queue
4. Discharge Patient by ID
5. See All Patients
6. See Urgent Patients
0) EXIT
Choice: 6
Urgent Patients
PATIENT ID: 6   NAME: Gabriela  AGE: 20 DOCTOR TO VISIT: 1      CAUSE: appendicitis     URGENCY: 1
PATIENT ID: 7   NAME: Santiago  AGE: 22 DOCTOR TO VISIT: 1      CAUSE: covid    URGENCY: 1
PATIENT ID: 4   NAME: raul      AGE: 30 DOCTOR TO VISIT: 2      CAUSE: brokenarm        URGENCY: 1
PATIENT ID: 17  NAME: Sara      AGE: 26 DOCTOR TO VISIT: 3      CAUSE: labor    URGENCY: 1
PATIENT ID: 1   NAME: dani      AGE: 22 DOCTOR TO VISIT: 2      CAUSE: allergy  URGENCY: 1
```