

# 1. Problema

## 1.1. Ejercicio 1

Una empresa de servicios guarda en una lista las tareas que debe realizar cada empleado. La estructura es la siguiente:

- Código Empleado
- Cantidad Tareas
- Cola de Tareas

La cola de tareas tiene la siguiente estructura:

- Área solicitante
- Descripción

Realizar procedimientos para este TDA que:

- Permita ingresar una nueva tarea en el empleado que tenga menos tareas.
- Permita ingresar una nueva tarea por código del empleado.
- Muestre el empleado con la mayor Cantidad de tareas y sus tareas pendientes.
- Muestre el empleado con la menor cantidad de tareas.

## 1.2. Ejercicio 2

En un hospital los pacientes sacan citas vía telefónica para ser atendidos en las diferentes especialidades que ofrece el hospital. Cada especialidad puede atender a 20 pacientes como máximo y la asignación de turnos es asignado por el tipo de gravedad.

Establezca el TDA Lista para cada especialidad y dentro de ellas, colas de prioridad. (La asignación de prioridad se debe hacer aleatoriamente).

- Asigne especialidades.
- Asigne pacientes a cada especialidad.
- Muestre los pacientes de una determinada especialidad.
- Lista la relación de todos los pacientes por especialidad.

## 2. Código

### 2.1. Ejercicio 1

#### 2.1.1. ColaTareas.h

```
#ifndef COLATAREAS.H
#define COLATAREAS.H
#include "iostream"

using namespace std;

class ColaTareas
{
public:
    class Tarea{
    public:
        Tarea();
        Tarea(int, string);
        Tarea * siguiente;
        Tarea * atras;
        int areaSolicitante;
        string descripcion;
    };
    ColaTareas();
    void add(int, string);
    void printTareas();
    void pop();
    Tarea * back();
    virtual ~ColaTareas();
protected:
private:
    Tarea * inicio;
    Tarea * cabeza;
};

void ColaTareas::printTareas() {
    if(!inicio){
        cout<<"Este emplado no tiene asignada ←
ninguna tarea"<<endl;
        return;
    }
    Tarea * temp = cabeza;
    while(temp){
        cout<<"Area Solicitante: "<<temp->areaSolicitante<<endl;
        cout<<"Descripcion: "<<temp->descripcion<<endl;
        temp = temp->atras;
    }
    cout<<endl;
}

void ColaTareas::pop() {
    if(!inicio) return;
    auto temp = cabeza;
    if(inicio == cabeza){
        inicio = nullptr;
        cabeza = nullptr;
    }
    else{
        cabeza = cabeza->atras;
        cabeza->siguiente = nullptr;
    }
    delete(temp);
}

void ColaTareas::add(int area, string des){
    Tarea * nuevo = new Tarea(area, des);
    if(!inicio){
        inicio = nuevo;
        cabeza = nuevo;
    }
    else{
        nuevo->siguiente = inicio;
        inicio->atras = nuevo;
        inicio = nuevo;
    }
}

ColaTareas::Tarea::Tarea() {
    areaSolicitante = -1;
    descripcion = nullptr;
    siguiente = nullptr;
    atras = nullptr;
}
```

```

}

ColaTareas::Tarea::Tarea(int area, string ←
descripcion){
    areaSolicitante = area;
    this->descripcion = descripcion;
    siguiente = nullptr;
    atras = nullptr;
}

ColaTareas::ColaTareas(){
    cabeza = nullptr;
    inicio = nullptr;
}

ColaTareas::~~ColaTareas(){
}

#endif // COLATAREAS.H
```

#### 2.1.2. ListaEmpleados.h

```
#ifndef LISTAEMPLEADOS.H
#define LISTAEMPLEADOS.H
#include "ColaTareas.h"
#include "iostream"

using namespace std;

class ListaEmpleados
{
public:
    class Empleado{
    public:
        Empleado();
        Empleado(int, string);
        string nombre;
        int codEmpleado;
        int cantidadTareas;
        ColaTareas tareas;
        Empleado * siguiente;
    };
    ListaEmpleados();
    void addEmpleado(string);
    void addTarea(int, string);
    void addTarea(int, int, string);
    void printMax();
    void printMin();
    bool find(int, Empleado *&);
    Empleado * min();
    Empleado * max();
    virtual ~ListaEmpleados();
protected:
private:
    Empleado * inicio;
    Empleado * fin;
    int tam;
};

void ListaEmpleados::printMin() {
    if(!inicio){
        cout<<"No existe ningun empleado"<<endl;
        return;
    }
    Empleado * menor = this->min();
    cout<<"EL empleado con menos tareas es: "<<menor->nombre<<endl;
    cout<<"Tareas:"<<endl;
    menor->tareas.printTareas();
    cout<<endl;
}

void ListaEmpleados::printMax() {
    if(!inicio){
        cout<<"No existe ningun empleado"<<endl;
        return;
    }
    Empleado * mayor = this->max();
    cout<<"EL empleado con mas tareas es: "<<mayor->nombre<<endl;
    cout<<"Tareas:"<<endl;
    mayor->tareas.printTareas();
    cout<<endl;
}
```

```

void ListaEmpleados::addTarea(int cod, int area, ←
    string des){
    Empleado * empleado;
    if(!find(cod,empleado))return;
    empleado->tareas.add(area,des);
    empleado->cantidadTareas += 1;
}

bool ListaEmpleados::find(int cod, Empleado *& ←
    empleado){
    if(!inicio)return false;
    empleado = inicio;
    while(empleado){
        if(empleado->codEmpleado == cod)return ←
            true;;
        empleado = empleado->siguiente;
    }
    return false;
}

void ListaEmpleados::addTarea(int area, string des←
    ){
    if(!inicio)return;
    Empleado* empleado = min();
    empleado->tareas.add(area,des);
    empleado->cantidadTareas += 1;
}

ListaEmpleados::Empleado* ListaEmpleados::max(){
    if(!inicio)return nullptr;
    Empleado * mayor = inicio;
    Empleado * temp = inicio->siguiente;
    while(temp){
        if(mayor->cantidadTareas < temp->←
            cantidadTareas){
            mayor = temp;
        }
        temp = temp->siguiente;
    }
    return mayor;
}

ListaEmpleados::Empleado* ListaEmpleados::min(){
    if(!inicio)return nullptr;
    Empleado * menor = inicio;
    Empleado * temp = inicio->siguiente;
    while(temp){
        if(menor->cantidadTareas > temp->←
            cantidadTareas){
            menor = temp;
        }
        temp = temp->siguiente;
    }
    return menor;
}

void ListaEmpleados::addEmpleado(string nombre){
    Empleado * nuevo = new Empleado(tam, nombre);
    tam++;
    if(!inicio){
        inicio = nuevo;
        fin = nuevo;
    }
    else{
        fin->siguiente = nuevo;
        fin = nuevo;
    }
}

ListaEmpleados::Empleado::Empleado(){
    codEmpleado = -1;
    cantidadTareas = 0;
    nombre = nullptr;
    siguiente = nullptr;
}

ListaEmpleados::Empleado::Empleado(int cod, string←
    nombre){
    codEmpleado = cod;
    this->nombre = nombre;
    cantidadTareas = 0;
    siguiente = nullptr;
}

ListaEmpleados::ListaEmpleados(){
    inicio = nullptr;
    tam = 0;
}

ListaEmpleados::~~ListaEmpleados(){}

#endif // LISTAEMPLEADOS.H

```

### 2.1.3. main.h

```

#include <iostream>
#include "ListaEmpleados.h"

enum Areas{ID, INFORMATICA, CONTABILIDAD, ←
    TESORERIA};

using namespace std;

int main()
{
    ListaEmpleados empleados;
    empleados.addEmpleado("Chris");
    empleados.addEmpleado("Nicoll");
    empleados.addEmpleado("Carlos");
    empleados.addTarea(0,INFORMATICA, "Tarea 1 ←
        para Chris");
    empleados.addTarea(0, CONTABILIDAD, "Tarea 2 ←
        para Chris");
    empleados.addTarea(0, TESORERIA, "Tarea 3 para←
        Chris");
    empleados.addTarea(1, TESORERIA, "Tarea 1 para←
        Nicoll");
    empleados.addTarea(1, ID, "Tarea 2 para Nicoll←
        ");
    empleados.addTarea(ID, "Tarea para el que ←
        tiene menos tareas");
    empleados.printMax();
    empleados.printMin();
}

```

## 2.2. Ejercicio 2

### 2.2.1. ColaPrioridad.h

```
#ifndef COLAPRIORIDAD_H
#define COLAPRIORIDAD_H
#include "vector"
#include <algorithm>

using namespace std;

class ColaPrioridad
{
public:
    class Paciente{
    public:
        Paciente();
        Paciente(string, int);
        string nombrePaciente;
        int gravedad;
    };
    ColaPrioridad();
    int size();
    void print();
    virtual ~ColaPrioridad();
    void insertar(string, int);
    Paciente* pop();
    Paciente* front();
protected:
private:
    vector<Paciente*> cola;
};

ColaPrioridad::Paciente * ColaPrioridad::front() {
    return cola.front();
}

int ColaPrioridad::size() {
    return cola.size();
}

void ColaPrioridad::print() {
    for(int i = 0; i < cola.size(); i++){
        cout<<cola[i]->nombrePaciente<<"-";
    }
    cout<<endl;
}

ColaPrioridad::Paciente * ColaPrioridad::pop() {
    if(cola.empty()) return nullptr;
    auto resultado = cola.front();
    int tam = cola.size() - 1;
    int pos = 0;
    cola[0] = cola[tam];
    cola.pop_back();
    tam--;
    while(pos <= tam) {
        if(2 * pos + 1 > tam) return resultado;
        if(2 * pos + 2 > tam) {
            if(cola[pos]->gravedad < cola[2 * pos + 1]->gravedad) {
                swap(cola[pos], cola[2 * pos + 1]);
                pos = 2 * pos + 1;
            }
        }
        if(cola[pos]->gravedad < cola[2 * pos + 1]->gravedad or cola[pos]->gravedad < cola[2 * pos + 2]->gravedad) {
            if(cola[2 * pos + 1]->gravedad > cola[2 * pos + 2]->gravedad) {
                swap(cola[pos], cola[2 * pos + 1]);
                pos = 2 * pos + 1;
            }
            else {
                swap(cola[pos], cola[2 * pos + 2]);
                pos = 2 * pos + 2;
            }
        }
        else {
            break;
        }
    }
    return resultado;
}
```

```
void ColaPrioridad::insertar(string nombre, int gravedad) {
    if(cola.size() == 20) {
        cout<<"No se pueden ingresar mas de 20 pacientes"<<endl;
        return;
    }
    cola.push_back(new Paciente(nombre, gravedad));
    int pos = cola.size() - 1;
    while(pos > 0) {
        if(cola[pos]->gravedad > cola[(pos - 1) / 2]->gravedad) {
            swap(cola[pos], cola[(pos - 1) / 2]);
            pos = (pos - 1) / 2;
        }
    }
}

ColaPrioridad::Paciente::Paciente(string nombre, int gravedad) {
    nombrePaciente = nombre;
    this->gravedad = gravedad;
}

ColaPrioridad::Paciente::Paciente() {
    nombrePaciente = nullptr;
    gravedad = 0;
}

ColaPrioridad::~ColaPrioridad() {
}

ColaPrioridad::ColaPrioridad() {
}

#endif // COLAPRIORIDAD_H
```

### 2.2.2. ListaEspecialidad.h

```
#ifndef LISTAESPECIALIDAD_H
#define LISTAESPECIALIDAD_H
#include "ColaPrioridad.h"

class ListaEspecialidad
{
public:
    class Especialidad {
    public:
        Especialidad();
        Especialidad(string);
        string nombreEspecialidad;
        ColaPrioridad* pacientes;
        Especialidad* siguiente;
    };
    ListaEspecialidad();
    void insertar(string);
    void printAll();
    void print();
    void mostrarPacientes(string);
    bool find(string, Especialidad*&);
    void agregarPaciente(string, string, int);
    virtual ~ListaEspecialidad();
protected:
private:
    Especialidad* inicio;
    Especialidad* fin;
};

void ListaEspecialidad::printAll() {
    if(!inicio) return;
    Especialidad* temp = inicio;
    while(temp) {
        cout<<"Pacientes de "<<temp->nombreEspecialidad<<" : "<<endl;
        ColaPrioridad pTemp = temp->pacientes;
        auto a = pTemp.size();
        for(int i = 0; i < a; i++){
            cout<<pTemp.pop()->nombrePaciente<<"<<endl;
        }
        temp = temp->siguiente;
    }
}
```

```

void ListaEspecialidad::mostrarPacientes(string <-
especialidad){
Especialidad * temp;
if(!find(especialidad, temp)){
cout<<"La especialidad no existe"<<endl;
return;
}
ColaPrioridad pTemp = temp->pacientes;
auto a = pTemp.size();
for(int i = 0; i < a; i++){
cout<<pTemp.pop()->nombrePaciente<<endl;
}
}

void ListaEspecialidad::print(){
Especialidad * temp = inicio;
while(temp){
cout<<temp->nombreEspecialidad<<endl;
temp = temp->siguiente;
}
}

void ListaEspecialidad::agregarPaciente(string <-
nombre, string especialidad, int gravedad){
Especialidad * temp;
if(!this->find(especialidad, temp))return;
temp->pacientes.insertar(nombre, gravedad);
}

bool ListaEspecialidad::find(string nombre, <-
Especialidad *&especialidad){
especialidad = inicio;
while(especialidad){
if(especialidad->nombreEspecialidad == <-
nombre)return true;
especialidad = especialidad->siguiente;
}
return false;
}

void ListaEspecialidad::insertar(string nombre){
Especialidad * nuevo = new Especialidad(nombre,<-
);
if(inicio == nullptr){
inicio = nuevo;
fin = nuevo;
}
else{
fin->siguiente = nuevo;
fin = nuevo;
}
}

ListaEspecialidad::Especialidad::Especialidad(<-
string nombre){
siguiente = nullptr;
nombreEspecialidad = nombre;
}

ListaEspecialidad::Especialidad::Especialidad(){
siguiente = nullptr;
nombreEspecialidad = nullptr;
}

ListaEspecialidad::~ListaEspecialidad(){
}

ListaEspecialidad::ListaEspecialidad(){
inicio = nullptr;
fin = nullptr;
}

#endif // LISTAESPECIALIDAD_H

```

```

int main()
{
srand (time(NULL));
ListaEspecialidad especialidades;
especialidades.insertar("Psicologia");
especialidades.insertar("Pediatria");
especialidades.insertar("Medicina General");
especialidades.agregarPaciente("Chris", "<-
Psicologia", rand() %10 + 1);
especialidades.agregarPaciente("Nicoll", "<-
Medicina General", rand() %10 + 1);
especialidades.agregarPaciente("Juan", "<-
Psicologia", rand() %10 + 1);
especialidades.agregarPaciente("Carlos", "<-
Pediatria", rand() %10 + 1);
especialidades.agregarPaciente("Ruben", "<-
Psicologia", rand() %10 + 1);
especialidades.mostrarPacientes("Psicologia");
cout<<endl;
especialidades.printAll();
}

```

### 2.2.3. main.h

```

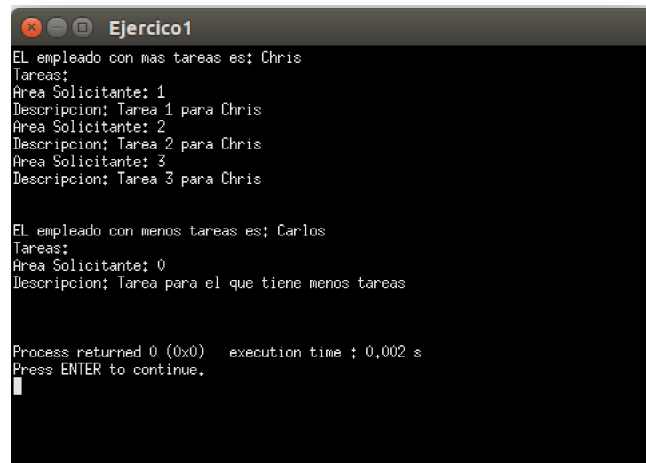
#include <iostream>
#include <algorithm>
#include "vector"
#include "ListaEspecialidad.h"
#include "random"
#include "time.h"

using namespace std;

```

## 3. Ejemplos

### 3.1. Ejercicio 1



```

Ejercicio1
El empleado con mas tareas es: Chris
Tareas:
Area Solicitante: 1
Descripcion: Tarea 1 para Chris
Area Solicitante: 2
Descripcion: Tarea 2 para Chris
Area Solicitante: 3
Descripcion: Tarea 3 para Chris

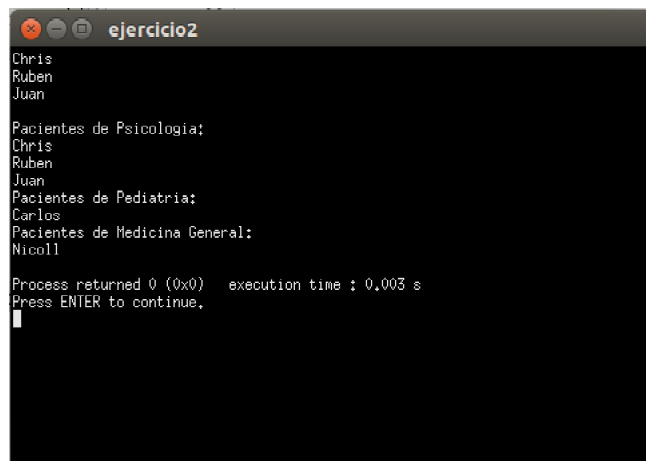
El empleado con menos tareas es: Carlos
Tareas:
Area Solicitante: 0
Descripcion: Tarea para el que tiene menos tareas

Process returned 0 (0x0)   execution time : 0,002 s
Press ENTER to continue.

```

Figura 1: Ejemplo Eje. 1

### 3.2. Ejercicio 2



```

ejercicio2
Chris
Ruben
Juan

Pacientes de Psicologia:
Chris
Ruben
Juan
Pacientes de Pediatria:
Carlos
Pacientes de Medicina General:
Nicoll

Process returned 0 (0x0)   execution time : 0,003 s
Press ENTER to continue.

```

Figura 2: Ejemplo Eje. 2