gonzz_
www.wuolah.com/student/gonzz_

RedMetrocpp.pdf

Examen ordinaria 2018 RESUELTO

1° Metodología de la Programación

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

```
#include "RedMetro.h"

RedMetro:: RedMetro(){
	this->num_lineas= 0;
	lineas= 0;
}

RedMetro:: ~RedMetro(){
	LiberaMemoria();
}

RedMetro:: RedMetro(const RedMetro & red){
	*this= red;
}

void RedMetro:: CopiaLineas(const RedMetro & red, int tam){
	for(int i= 1; i <= tam; i++){
		(*this)[i]= red[i];
	}
}

bool RedMetro:: EstaVacia()const{
	return lineas== 0;
}

void RedMetro:: LiberaMemoria(){
	if (!EstaVacia()){
		delete [] lineas;
		lineas= 0;
	}
}

void RedMetro:: ReservaMemoria(int tam){
	assert (tam > 0);
	lineas= new Linea[tam];
	num_lineas= tam;
}

RedMetro & RedMetro:: operator= (const RedMetro & red){
	if (&red != this){
		if(num_lineas!= red.num_lineas){
			if(lineas!= 0)
				LiberaMemoria();
			ReservaMemoria(red.num_lineas);
		}
		CopiaLineas(red, num_lineas);
	}

	return *this;
```

```
}

RedMetro & RedMetro:: operator+=(const Linea & linea){
	RedMetro nueva;
	nueva.ReservaMemoria(num_lineas+1);
	if(num_lineas>0)
		nueva.CopiaLineas(*this, num_lineas);
	nueva[num_lineas+1]=linea;

	*this= nueva;

	return *this;
}

int RedMetro::GetNumLineas()const{
	return num_lineas;
}

bool RedMetro:: IndiceValido(int i)const{
	return i > 0 && i <= num_lineas;
}

Linea & RedMetro:: operator[](int i){
	assert(IndiceValido(i));
	return lineas[i-1];
}

Linea RedMetro:: operator[](int i) const{
	assert(IndiceValido(i));
	return lineas[i-1];
}

int RedMetro::GetNumTotalParadas()const{
	int total= 0;
	for (int i= 1; i <= num_lineas; i++){
		for (int j= 1; j <= (*this)[i].GetNumParadas(); j++){
			if((*this)[i][j].GetIndice()>total)
				total= (*this)[i][j].GetIndice();
		}
	}
	return total;
}

ostream & operator<<(ostream & flujo, const RedMetro &red){
	flujo<< "Número de líneas: " << red.GetNumLineas() << endl;
	for(int i= 1; i <= red.GetNumLineas(); i++)
		flujo<< red[i];
	return flujo;
}
```

```
istream & operator>>(istream & flujo, RedMetro &red){
	int num_lineas;
	flujo >> num_lineas;

	assert(num_lineas>= 0);

	RedMetro nueva_red;

	if(num_lineas > 0){
		int num_paradas;

		while(flujo>> num_paradas){
			Linea nueva_linea;

			for(int i= 0; i < num_paradas; i++){
				int indice_parada;
				char activac;
				bool activa;
				flujo >> indice_parada >> activac;
				(activac=='S')? activa= true : activa= false;
				InfoParada parada(activa, indice_parada);
				nueva_linea+=parada;
			}
			nueva_red+=nueva_linea;
		}
	}
	red= nueva_red;
	return flujo;
}

RedMetro:: RedMetro(char * cadena){
	lineas= 0;
	num_lineas= 0;
	ifstream fi(cadena);
	string metro;
	if (fi){
		getline(fi, metro);
		if(metro=="METRO"){

			int num_lineas;
			fi >> *this;
		}
	}
	fi.close();
}

int RedMetro:: ParadasActivas(const RedMetro &red){
	int paradas_activas= 0;
```

```
	for(int i= 1; i <= num_lineas; i++)
		paradas_activas+=red[i].GetNumParadasActivas();

	return paradas_activas;
}

double RedMetro:: CalculaCalidad(const RedMetro &red){
	return num_lineas*0.3+ParadasActivas(*this)*0.7;
}

bool RedMetro:: operator==(const RedMetro &red){

	return CalculaCalidad(*this)==CalculaCalidad(red);
}

bool RedMetro:: operator!=(const RedMetro &red){
	return !((*this)==red);
}

bool RedMetro:: operator>(const RedMetro &red){
	return CalculaCalidad(*this) > CalculaCalidad(red);
}

int RedMetro:: MejorConectada()const{
	assert(!EstaVacia());
	int total_paradas= GetNumTotalParadas();
	int * contador_paradas= new int [total_paradas];

	for(int i= 0; i < total_paradas; i++){
		contador_paradas[i]= 0;
	}

	for(int i= 1; i <= num_lineas; i++){
		int paradas_linea= (*this)[i].GetNumParadas();
		for(int j= 1; j <= paradas_linea; j++){
			int parada_actual= (*this)[i][j].GetIndice();
			contador_paradas[parada_actual-1]++;
		}
	}

	int mejor_conectada= 0, conexiones_max= contador_paradas[0];
	for(int i= 0; i < total_paradas; i++){
		if(contador_paradas[i]> conexiones_max){
			conexiones_max= contador_paradas[i];
			mejor_conectada= i;
		}
	}

	delete [] contador_paradas;
```

```
        contador_paradas= 0;

        return mejor_conectada+1;
}

bool RedMetro:: EstaTotalmenteOperativa(int i)const{
        assert(IndiceValido(i));

        bool operativa= true;

        for(int j= 1; j <= (*this)[i].GetNumParadas() && operativa; j++){
                operativa= (*this)[i][j].EstaActiva();
        }

        return operativa;
}
```