Relación de Problemas: STL.

1.Definir una función que permita invertir un objeto de tipo list. Los elementos que contiene la lista son enteros.

Invertir(const list<int> & lsource, list<int> & ldestino)

```
#include<list>
using namespace std;
//Suponiendo que la lista destino tiene elementos y queremos sustituirlos por los elementos de la lista actual.
void Invertir( const list<int> & lsource, list<int> & lddestino){
	//En c++11 y c++14 podemos hacer auto ritsource y auto itdestino
	list<int>::const_reverse_iterator ritsource = lsource.crbegin();
	list<int>::iterator itdestino = ldestino.begin();

	for( ;ritsource != lsource.crend();++ritsource){
		*itdestino = *ritsource;
		++itdestino;
	}
}

//Suponemos que la lista destino está vacía
void Invertir( const list<int> & lsource, list<int> & lddestino){
	list<int>::const_reverse_iterator ritsource = lsource.crbegin();

	for(;ritsource != crend(); ++ritsource){
		ldestino.push_front(*ritsource);
	}
}
```

2.Suponer que tenemos información de los alumnos que desean acceder a una carrera junto con su nota de selectividad.
Definir una función que obtenga una cola de prioridad (priority_queue) con la información de todos los alumnos, de manera que la prioridad se define de mayor a menor valor de selectividad. Así la función sería:

void ObtenerPrioridad (const list<alumno>& alumnos, priority_queue<alumno>& pq) alumnos almacena la información de todos los alumnos;

Para saber que alumno tiene mayor prioridad, debemos sobrecargar el operator>

```
bool operator>(const alumno & otroAlumno){
	return *(this).nota_selectividad > otroAlumno.nota_selectividad;
}

#include <queue>
using namespace std;
void ObtenerPrioridad (const list<alumno>& alumnos, priority_queue<alumno>& pq){
	list<int>::auto it = alumnos.cbegin();//Posible uso en c++11
	for(;it != alumnos.cend();++it){
		pq.push(*it);
	}
}
```

3.Dada la clase list instanciada a enteros, crear una función que elimine los elementos pares de la lista. Para implementarla hacer uso de los iteradores.

```
#include <list>
using namespace std;

void eliminarElementosPares(list<int> & mylist){
	list<int>::iterator it;

	for(it = mylist.begin();it != mylist.end();++it){
		if(*it % 2 == 0)
			it = mylist.erase(it);
	}
}
```

4.Definir el T.D.A DoblePila que contiene dos pilas de caracteres usando un único vector para ello. Así una de las pilas empieza a poner sus datos desde la posicion 0 hacia adelante y la otra desde el máximo espacio reservado hacia atrás. Un esquema de este tipo se puede ver en la siguiente figura.

Dar una representación de este TDA e implementar los siguientes métodos:

1. char Tope (key_pila kp): devuelve el tope de la pila con código kp. Key_pila puede ser un enumerado con dos valores Pila1 y Pila2.
2. Void Poner (key_pila kp, char c): inserta un nuevo elemento en el tope de la pila kp. Si ya no tenéis espacio sobre el vector deberéis redimensionar el vector.
3. Void Quitar(key_pila kp): elimina del tope de la pila kp.
4. Bool Vacia(key_pila kp): indica si la pila kp está vacía.

```
Class DoblePila{
private:
	vector<char> datos;

public:
	char Tope(key_pila kp){
		if(kp == “Pila1”){
			vector<char>::iterator it;
			for(it = datos.begin(); *it != '0' || it != datos.end(); ++it);
			return *it;
		}
		if(kp == “Pila2”){
			vector<char>::reverse_iterator it;
			for(it = datos.rbegin(); *it != '0' || it != datos.rend(); ++it);
			return *it;
		}
	}

	void Poner(key_pila kp, char c){
		if ( Tope( “Pila1” ) == Tope ( “Pila2” ){
			char ultimo = datos[ datos.size() - 1];
			char tope = Tope( “Pila2” );
			datos.resize( 2* datos.size() );
```

```
                vector<char>::reverse_iterator rever = datos.rbegin();
                vector<char>::reverse_iterator pila;
                for(pila = datos.rbegin(); *pila != tope;++pila);
                for(; *pila != *rever;--pila && ++rever){
                        *rever = *pila;
                }
        }
        if(kp == “Pila1”){
                vector<char>::iterator it;
                for(it=datos.begin(); it != Tope(kp);++it);

                ++it;
                it = datos.insert(it, c);
        }
        if(kp == “Pila2”){
                vector<char>::reverse_iterator it;
                for(it=datos.rbegin(); it != Tope(kp);++it);

                ++it;
                it = datos.insert(it, c);
        }
}

void Quitar(key_pila kp){
        if(kp == “Pila1”){
                vector<char>::iterator it;
                for(it=datos.begin(); it != Tope(kp);++it);
                ++it;
                it = datos.erase(it);
        }
        if(kp == “Pila2”){
                vector<char>::reverse_iterator it;
                for(it=datos.rbegin(); it != Tope(kp);++it);
                ++it;
                it = datos.erase(it);
        }
}

bool Vacia(key_pila kp){
        if(kp == “Pila1”)
                return datos[0] == '0';

        if(kp == “Pila2”)
                return datos[datos.size() -1] == '0';
}
};
```

5.Suponer que tenemos el T.D.A Matriculas que almacena parejas de dni de un alumno y el código de asignatura en el que está matriculado. Escoger la representación más adecuada para este tipo de dato, de forma que si consulta por los alumnos que están matriculados en una asignatura concreta sea lo más eficiente posible. Y si además, pregunto por las asignaturas que un determinado alumno está matriculado también sea

eficiente. Implementar el método insertar, borrar que añade una nueva matrícula y borra una matrícula existente.

```
#include <map>
using namespace std;
Class Matriculas{
private: // first es el string perteneciente al dni de un alumno y el second es es string perteneciente al código de la asignatura.
        multimap<string,string> matriculas;
};
Void Matriculas::Insertar(const string& dni, const string& cod_asig){
        matriculas.insert( pair<string,string>(dni, cod_asig) );
        //A partir de c++11 se puede utilizar make_pair( dni, cod_asig );
}


Void Matricuals::Borrar(const string& dni, const string& cod_asig){
        multimap<string,string>::iterator it;
        it = matriculas.find(dni);
        for(; (*it).first == dni;++it){
                if( (*it).second == cod_asig)
                        it = matriculas.erase(it);
        }
}
```

6.Siguiendo con el T.D.A Matriculas del ejercicio anterior:

1. Definir un iterador que permita iterar sobre todas las matrículas.
2. Implementar el método que permite consultar todos los alumnos (dni) matriculados en una asignatura:

list<string> Matriculas::GetAlumnos(const string& cod_asig);

3. Implementar el método que permite consultar todos las asignaturas (cod_asig) en las que está matriculado un alumno:

list<string> Matriculas::GetAsignaturas(const string& dni);

1)

```
class iterator{
private:
        multimap<string,string>::iterator it;
public:
        iterator operator++(){
                string dni = *it.first;
                for(;*it.first == dni;++it);
                return *this;
        }
        iterator operator--(){
                string dni = *it.first;
                for(;*it.first == dni;--it);
                return *this;
        }
        pair<string,string> operator*(){
                return *it;
        }
```

```
        bool operator==( const iterator& i)const{
                return i.it == it;
        }
        bool operator!=( const iterator& i)const{
                return i.it != it;
        }

        friend class Matriculas;
};
```

2)

```
list<string> Matriculas::GetAlumnos( const string& cod_asig){
        list<string> asignatura;

        Matriculas::iterator alumnos;
        multimap<string,string>::iterator it;

        for(alumnos = matriculas.begin(); alumnos != matriculas.end();++alumnos){
                for(it = alumnos; it != alumnos + 1;++it){
                        if( (*it).second == cod_asig)
                                aisgnatura.push_back( (*it).second );
                }
        }

        return asignatura;
}
```

3)

```
list<string> Matriculas::GetAsignaturas( const string& dni){
        list<string> asignaturas;
        Matriculas::iterator alumno;
        multimap<string,string>::iterator it;
        alumno = matriculas.find( dni );
        it = alumno;
        ++alumno;

        for(;it != alumno;++it){
                asignaturas.push_back( (*it).second);
        }
        return asignaturas;
}
```