# Relación 3. – STL. Alberto Rodríguez Santana

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

```
void Invertir(const list<int> & lsource, list<int>& ldestino){
        list<int>::const_iterator it;
        for(it=lsource.begin(); it!=lsource.end();it++)
        ldestino.insert(ldestino.begin(),*it);
}
```

**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: (alumnos almacena la información de todos los alumnos).

```
struct alumno{
        char dni[9];
        string nombre;
        string apellidos;
        string correo;
        double nota_selectividad;
};

struct Comp{
        bool operator<(const alumno& a1, const alumno& a2){
                return a1.nota_selectividad<a2.nota_selectividad;
        }
};

void ObtenerPrioridad (const list<alumno> & alumnos, prority_queue<alumno> & pq){
        priority_queue<alumno, list<alumno>,Comp> mypq;
        while(!alumnos.empty())
                mypq.push(alumno);
        while(!mypq.empty ())
                pq.push(alumno);
}
```

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

```
void EliminaPares(list<int>& L){
        list<int>::iterator it;
        while(it!=L.end()){
                if((*it)%2==0)
                        it=L.erase(it);
                else
                        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 posición 0 hacia adelante y la otra desde el máximo espacio reservado hacia atrás.
Dar una representación de este TDA e implementar los siguiente 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.

```
#include <vector>
enum key_pila{Pila1, Pila2};

class DoblePila{
        private:
                vector<char> v;
                int nelem1;
                int nelem2;
        public:
                DoblePila(){
                        nelem1=0;
                        nelem2=0;
                }

1.              char Tope(key_pila kp){
                        vector<char>iterator it;
                        if(kp==Pila1){
                                it=v.begin()+nelem1-1;
                                return *it;
                        }
                        else{
                                it=v.end()-nelem2+1;
                                return *it;
                        }
                }  //fin funcion Tope

2.              void Poner(key_pila kp,char c){
                        vector<char>iterator it;
                        if(kp==Pila1){
                                if(v.empty() || nelem2==0){
                                        v.push_back(c);
                                         nelem1++;
                                }
                                else{
                                        if((nelem1+nelem2)>=(v.size()/2)){
                                                v.resize(v.size()*2);
                                                it=v. begin()+nelem1;
                                                v.insert(it,c);
                                                nelem1++;
                                        }
                                        else{
```

```
                                        it=v. begin()+nelem1;
                                        v.insert(it,c);
                                        nelem1++;
                                }
                        }
                }
                else{
                        if(v.empty()){
                                v.push_back(c);
                                 nelem2++;
                        }
                        else{
                                if((nelem1+nelem2)>=(v.size()/2)){
                                        v.resize(v.size()*2);
                                        it=v. end()-nelem2;
                                        v.insert(it,c);
                                        nelem2++;
                                }
                                else{
                                        it=v.end()-nelem2;
                                        v.insert(it,c);
                                        nelem2++;
                                }

                        }
                }
        }  //Fin funcion Poner
```

3.

```
        void Quitar(key_pila kp){
                if(kp==Pila1){
                        assert(nelem1>0);
                        nelem1--;
                }
                else{
                        assert(nelem2>0);
                        nelem2--;
                }

                if((nelem1+nelem2)<(v.size()/4)){
                        v.resize(v.size/2);
                }
        }  //Fin funcion Quitar
```

4.

```
        bool Vacia(key_pila kp){
                if(kp==Pila1)
                        return nelem1==0;
                else
                        return nelem2==0;
        }  //Fin funcion Vacia

};
```