```
//LISTA CIRCULAR
//ejemplo sencillo de listas circulares
//este programa solo intenta mostrar el comportamiento de los punteros en las listas circulares
//no es robusto, hay que agregar excepciones y otros métodos


#include<iostream>
using namespace std;


template <class T> class ListaC ;

template<class T>
class Nodo
 {
    friend class ListaC <T>;
    public:
                        Nodo(const T& _v):_info(_v),_sig(0){}
    private:
        T _info;
        Nodo <T> *_sig;
  };


template <class T>
class ListaC
{
 public:
   ListaC( );  //constructor
           // Escribir el constructor copia como ej
   ~ListaC( );  //destructor
   void AltaPrin(const T &);  //alta al principio
   void AltaFin (const T &);  //alta al final
   void BajaPrin( );    //borra primer nodo
   void BajaFin( );     //borra ultimo nodo
   bool Vacia( ) const; //retorna true si lista vacia
   void Emitir ( ) const;  //emite la lista

 private:
   Nodo <T> * _principio;
};

template <class  T>
ListaC <T>::ListaC():_principio(0){}

template <class T>
ListaC<T>::~ListaC()
{
  if(_principio)
       {
         if (_principio->_sig == _principio)delete _principio;
         else
          {
            Nodo <T> *_aux1, *_aux2;
            _aux1=_principio->_sig;
            _principio->_sig =0;
```

```
            while(_aux1!=0)
              {
               _aux2=_aux1;
               _aux1=_aux1->_sig;
               delete _aux2;
              }
            }
          }
}


template <class T>
void ListaC <T> :: AltaPrin (const T & _v)
{
 Nodo<T> *_nuevo=new Nodo <T>(_v), *_aux;
 if(Vacia())
           {
            _principio=_nuevo;
            _nuevo->_sig=_nuevo;
            }
 else
   {
    _nuevo->_sig=_principio;
    _aux=_principio;
    while (_aux->_sig != _principio) _aux=_aux->_sig;
    _aux->_sig =_nuevo;
    _principio=_nuevo;
   }
}

template <class T>
void ListaC <T> :: AltaFin (const T & _v)
{
 Nodo<T> *_nuevo=new Nodo <T>(_v), *_aux ;
 if(Vacia())
 {
     _principio=_nuevo;
     _nuevo->_sig=_nuevo;
  }
 else
    {
     _aux=_principio;
     while (_aux->_sig != _principio) _aux=_aux->_sig;
     _aux->_sig=_nuevo;
     _nuevo->_sig=_principio;
    }
}


template <class T>
void ListaC <T> :: BajaPrin()
{
Nodo<T> *_aux1, *_aux2;
if (!Vacia())
 {
   if (_principio->_sig==_principio)
```

```
                    {
                     delete _principio;
                     _principio=0;
                     }
  else
     {
      _aux1=_principio;
      _aux2=_aux1->_sig;
      while (_aux2->_sig!=_principio)_aux2=_aux2->_sig;
      _aux2->_sig=_aux1->_sig;
      _principio =_principio->_sig;
      delete _aux1;
     }
 }

else cout<<"lista vacia"<<endl;
}



template <class T>
void ListaC <T> :: BajaFin()
{
 Nodo <T> *_aux1,*_aux2;
 if (!Vacia())
 {if (_principio->_sig==_principio)
                    {
                     delete _principio;
                     _principio=0;
                     }
  else
     if (_principio->_sig->_sig == _principio)
        {
         _aux1=_principio->_sig;
         _principio->_sig=_principio;
         delete _aux1;
        }
  else
     {
      _aux1=_principio->_sig;
      while (_aux1->_sig->_sig!=_principio)_aux1=_aux1->_sig;
      _aux2=_aux1->_sig;
      _aux1->_sig=_aux2->_sig;
      delete _aux2;
     }
 }
}

template <class T>
void ListaC <T> :: Emitir () const
{
 Nodo<T> *_aux=_principio;
if(!Vacia())
 { do
   { cout<<_aux->_info;
     _aux=_aux->_sig;
```

```
    }
 while(_aux!=_principio);
 }
}

template <class T>
bool ListaC <T> :: Vacia () const
{
  return (_principio==0);
}


void main()
{

ListaC <int> c1,c2;
c1.AltaPrin(4);
c1.AltaPrin(3);
c1.AltaFin(5);
c1.Emitir();
cout<<endl;
c2.AltaPrin(2);
c2.AltaPrin(1);
c2.AltaFin(3);
c2.Emitir();
cout<<endl;
c1.BajaPrin();
c1.Emitir();
cout<<endl;
c1.BajaFin();
c1.Emitir();


}
```